C Bourne Korn bash tcsh awk grep sed

실례를 통하여 배우는

UNIX쉘

김책공업종합대학출판사
주체91

실례를 통하여 배우는

UNIX 쉘

김책공업종합대학출판사

차　　례

제 8 장. 대화형Bourne쉘

제 9 장. C쉘

제 10 장. Korn쉘

제 11 장. 대화형bash쉘

제 12 장. bash쉘에 의한 프로그람작성법

제 13 장. 대화형TC쉘

머 리 말

쉘을 학습하는것은 아주 흥미 있다. 이 책은 실천을 통하여 쉘을 흥미 있고 유익하게 배울수 있도록 씌여 졌다. 첫판이 나온 다음 이 책의 도움을 받은 많은 사람들은 쉘프로그람작성이 결코 힘들지 않다고 말하고 있다. 실례를 통하여 배우면 아주 쉽고 흥미가 있다. 사실 이러한 적극적인 지원으로 하여 필자는 출판사로부터 2개의 인기 있는 쉘들인 Bash쉘과 TC쉘이 추가된 새로운 갱신판본을 집필할데 대한 부탁을 받게 되였다. 비록 Bash쉘과 TC쉘들이 보통 Linux쉘들과 련관되여 있지만 UNIX를 사용하는 사람들도 그것을 자유롭게 사용할수 있다. 오늘날 많은 UNIX사용자들은 이 쉘들이 실제로 프로그람작성능력이 개선되였을뿐아니라 유용한 대화환경을 제공해 주기때문에 전통적인 UNIX쉘들보다 이것들을 더 많이 사용하고 있다.

《실례를 통하여 배우는 UNIX쉘》을 집필한것은 쉘프로그람작성자들이 널리 사용하는 여러가지 쉘들과 UNIX편의프로그람을 위한 강의를 하고 개발한 필자의 19년간의 사업에서 최고절정으로 된다. 강의를 위한 과정안들은 캘리포니아 싼타크루즈종합대학, 캘리포니아다비스UNIX프로그람종합대학, 싼마이크로체계교육, 애플콤퓨터, 디안자대학과 전 세계의 수많은 판매자들에 의해 사용되고 있다. 사용자들의 요구에 따라서 보통 모든 쉘들을 한번에 가르치지 않고 한개씩 가르친다. 많은 사용자들의 수요를 충족시키기 위해 개별적인 UNIX쉘들과 도구들에 대한 참고서들을 만들었다.

《Grep, Sed 그리고 Awk》와《체계관리자를 위한 Bourne쉘》또는《대화형Korn쉘》을 가르칠 때 한 학생이 다음과 같은 질문을 하였다. 《모든 쉘들과 grep, sed 그리고 awk와 같은 중요한 편의프로그람들을 다같이 취급하는 책을 얻을수 있습니까? awk책을 얻고 또는 grep와 sed에 대한 책을 따로 얻어야 합니까? 그것들을 모두 서술한 한권의 책이 있습니까? 나는 쉘프로그람작성자가 되기 위해 3개 또는 네개의 책들을 사고 싶지 않습니다.》

그 대답으로 이러한 론점들을 개별적으로 서술한 수많은 훌륭한 책들과 그것들을 모두 서술한 몇권의 UNIX책들을 권고할수 있지만 학생들은 모든것을 다 포함하면서도 속성교재가 아닌 한권의 책을 요구하고 있다. 그들은 UNIX도구들, 정규식들, 3개의 쉘들, 인용부호찍기규칙들, 쉘들의 비교, 련습 등을 모두 서술한 한권의 책을 요구한다. 이 책이 바로 그런 책이다. 이 책을 쓸 때 강의를 어떻게 하며 장들을 어떻게 편성하겠는가에 대하여 생각해 보았다. 쉘프로그람작성 강의들에서 첫번째 론점은 보통 쉘이란 무엇이며 그것은 어떻게 동작하는가에 대하여 알려 주는것이다. 그다음에 쉘프로그람작성자의 도구칸에서 제일 중요한

도구들인 grep, sed 그리고 awk와 같은 UNIX편의프로그람들에 대하여 알려 주는것이다. 쉘을 배울 때 처음에는 모든것이 지령행에서 수행될수 있는 대화형프로그람과 같이 보이고 그다음에는 쉘스크립트들에서 프로그람작성구조들을 서술하고 설명하는 프로그람작성언어와 같이 보인다(C와 TC쉘들은 프로그람작성언어와 거의 같기때문에 대화형식의 사용방법을 서술하는 장들이 따로 있지만 프로그람작성구조들을 서술하는 장은 한개만 있다.). 쉘프로그람작성강의가 이틀 또는 한주일 지어는 한학기동안 진행된다 하더라도 그것이 끝나면 학생들은 스크립트를 능숙하게 작성할수 있으며 흥미를 가지게 된다. 이 과정에 그들은 쉘프로그람작성법을 배운다. 이 책은 독자들이 강의를 받던가 또는 자체로 학습할 때 쉘프로그람작성법에 대하여 가르쳐 준다. 리해를 더 쉽게 그리고 빨리 할수 있게 하는 간단한 실례들을 줌으로써 프로그람의 매 행에 대한 출력과 설명을 통해 개념들을 습득하게 하였다. 이 방법은 처음으로 집필한 책인 《실례를 통하여 배우는 실용추출 및 보고서작성(Perl)》을 학습한 사람들에게 아주 인기가 있었는데 지금은 《실례를 통하여 배우는 UNIX쉘》이 쉘프로그람을 작성하고 읽고 그리고 보수하려고 하는 사람들에게서 호평을 받고 있다.

5개의 쉘들이 병렬로 서술되므로 만일 방향바꾸기가 한개 쉘에서 어떻게 진행되는가를 알려고 할 때 다른 쉘들에 대한 매 장에서도 그 문제에 대한 똑같은 설명이 있으므로 그것들을 참고하여도 된다. 그것들을 빨리 비교해 보려면 이 책의 부록 2를 참고하면 된다. 지령들이 어떻게 동작하는가에 대해서 기억을 상기시킬 필요가 있는 어떤 특수한 지령에 대해서 충분한 정보를 알려고 할 때 다른 책이나 UNIX의 안내페지들을 참고하는것은 시끄러운 일이다. 시간을 절약하기 위해 부록 1에 쓸모가 많은 지령들 즉 그것들의 문법과 정의들에 대한 목록을 추가하였다. 보다 어렵고 자주 사용되는 지령들에 대해서는 실례와 설명들을 주었다.

부록 2에 있는 비교표는 특별히 스크립트들을 한 쉘에서 다른 쉘에로 넘길 때 서로 다른 쉘들을 직접 관리할수 있게 하며 지령이 동작하는 방식에 대해 상기할 필요가 있을 때 문법검사를 빨리 진행할수 있게 해준다. 쉘프로그람작성자들에게 있어서 가장 큰 애로의 하나는 인용부호를 정확히 사용하는것이다. 부록 3에 있는 인용부호찍기규칙에 대한 부분은 아주 복잡한 지령행들에서 인용부호를 성과적으로 찍기 위한 단계들을 하나하나 보여 준다. 이 과정은 인용부호들을 정확히 일치시켜 줌으로써 스크립트들을 오유수정할 때 프로그람작성자들이 헛되게 소비하는 시간을 줄인다. 독자들이 이 책을 가치 있는 참고서로 여기리라고 생각한다. 이 책의 목적은 실례를 통하여 설명하고 문제점들을 보다 간단히 하여 독자들이 흥미있게 배우고 시간을 절약할수 있게 하자는것이다. 책에는 강의들에서 이야기한 내용들이 그대로 실려 있으므로 독자들이 짧은 시간에 훌륭한 쉘프로그람작성자가 되리라고 확신한다. 바로 여기에 필요한 모든것이 준비되여 있기때문에 실례를 통해 쉘프로그람작성법을 배우는것은 흥미가 있으리라고 본다.

제 1 장. UNIX쉘(shell)에 대한 개념

제 1 절. 정의와 기능

쉘은 그림 1-1에서 보여 준바와 같이 사용자와 UNIX조작체계의 심장부인 핵심부 프로그람사이의 결합부로 사용되는 특정한 프로그람이다. 핵심부는 기동될 때 기억기에 적재되여 전원이 차단될 때까지 체계를 관리한다.

쉘은 프로쎄스를 작성하고 조종하며 기억기, 파일체계, 통신 등을 관리한다. 쉘프로그람을 비롯한 모든 프로그람들은 자기원판상에 존재한다. 핵심부는 이 프로그람들을 기억기에 적재시키고 실행하며 완료되면 그 체계를 지운다. 쉘은 사용자가 등록가입할 때 기동되는 편의프로그람이다. 이것은 지령행이나 스크립트파일에서 입력되는 지령들을 해석하여 사용자가 핵심부와 대화하도록 한다. 등록가입할 때 대화형쉘은 기동되여 사용자가 입력할것을 재촉한다. 한개의 지령을 입력했을 때 쉘의 기능은 다음과 같다.

ㄱ) 지령행을 구문해석한다.
ㄴ) 통용기호, 방향바꾸기, 파이프와 일감조종을 처리한다.
ㄷ) 지령을 탐색하고 찾으면 실행한다.

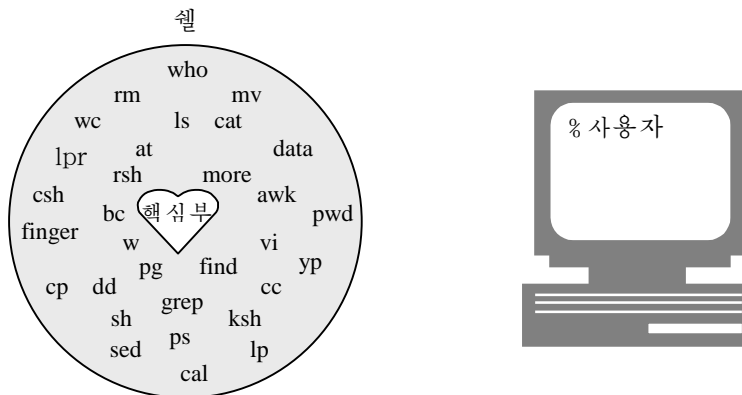



그림 1-1. 핵심부, 쉘, 사용자

처음 UNIX를 배울 때 재촉문상태로부터 지령을 입력하고 실행하는데 많은 시간을 소비하게 된다. 이때 사용자는 쉘을 대화형으로 사용한다.

만일 같은 지령묶음을 매번 입력하는 경우 과제들을 자동적으로 실행하도록 할수 있다. 이렇게 하려면 스크립트파일에 지령들을 써넣고 실행하면 된다. 쉘스크립트는 묶음파일과 같다. 즉 스크립트는 파일에 써넣어 진 UNIX지령들의 목록이고 파일이 입력되면 실행된다. 보다 정교한 스크립트들은 조건판별, 순환고리, 파일시험 등을 위한 프로그람작성구조들을 포함한다. 스크립트를 작성하려면 프로그람작성구조와 기술을 배워야 할뿐아니라 UNIX편의프로그람과 스크립트가 어떻게 동작하는가를 충분히 파악하여야 한다. 지령출력과 파일들의 조작을 위해 스크립트에서 사용되는 가장 위력한 도구들인 grep, sed, awk와 같은 편의프로그람들이 있다. 특수한 쉘에 대한 도구들과 프로그람작성구조를

알게 되면 유용한 스크립트들을 작성할수 있다. 스크립트내부로부터 지령을 실행할 때 사용자는 쉘을 프로그람작성언어처럼 사용한다.

1. 3개의 기본 UNIX 쉘

대부분의 UNIX체계에서 3개의 중요한 쉘들은 Bourne쉘(AT&T쉘), C쉘(Berkeley 쉘), Korn쉘(Bourne쉘 웃준위)들이다. 이 3개의 쉘들은 사용자와 대화형으로 실행될 때에는 같은 방식으로 동작하지만 스크립트작성언어로서 사용될 때에는 문법과 기능에서 일련의 차이가 있다.

Bourne쉘은 표준UNIX쉘이며 체계를 관리하기 위해 사용된다. rc start, stop스크립트들, shut down과 같은 대부분의 체계관리스크립트들은 Bourne쉘스크립트이고 단일사용자방식에서 뿌리(root)에서와 같이 실행될 때 체계관리자에 의해 사용되는 쉘이다. 이 쉘은 AT&T에서 작성되였는데 간결하고 압축되였으며 처리속도가 빠른것으로 알려 져 있다. Bourne쉘의 기정재촉문은 화페기호($)이다.

C쉘은 버클리(Berkeley)에 의해 개발되였으며 지령행리력, 별명, 내부산수연산, 파일이름완성, 일감조종 등과 같은 많은 특징들이 추가되였다. C쉘은 쉘들을 대화형으로 사용하는 사용자들에게 있어서 Bourne쉘보다 더 좋지만 관리자는 Bourne쉘스크립트가 C쉘로 쓴 스크립트보다 더 간결하고 빠르기때문에 스크립트작성에서 Bourne쉘을 자주 사용한다. C쉘의 기정재촉문은 퍼센트기호(%)이다.

Korn쉘은 AT&T에서 다비드 콘(David Korn)에 의해 작성된 Bourne쉘의 웃준위이다. 일련의 특징들이 추가되여 C쉘보다 더 강력한 기능들이 이 쉘에 있다. Korn쉘의 특징은 편집할수 있는 리력, 별명, 함수들, 정규식통용기호(regular expression wildcard), 내부산수연산, 일감조종, 협동프로쎄스, 특정한 오유수정수단 등을 포함한다. Bourne쉘은 Korn쉘과 거의 완전한 웃방향호환성을 가지므로 이전의 Bourne쉘프로그람도 이 쉘에서 실행할수 있다. Korn쉘의 기정재촉문은 화페기호 $이다.

2. Linux 쉘

Bash와 TC쉘을 흔히 《Linux》쉘이라고 부르지만 임의의 UNIX체계에서도 자유롭게 사용할수 있으며 번역할수 있다. 즉 실지로 쉘들은 Solaris 8과 Sun UNIX조작체계와 결합된다. Linux를 설치할 때 GNU쉘과 도구들은 호출할수 있으나 표준UNIX쉘과 도구들은 호출할수 없다.

Linux는 일련의 쉘들을 제공하지만 그가운데서 Bourne Again쉘(bash)과 TC쉘(tcsh)은 가장 인기 있는 쉘들이다. Z쉘은 Bourne Again쉘, TC쉘, Korn쉘들의 일부 특징들을 통합한 또 다른 linux쉘이다. Korn쉘의 파생인 공개령역 Korn쉘 PDKSH(Public Domain Korn Shell)을 사용할수 있으며 값을 치르고 대부분의 알려 지지 않은 보다 작은 쉘들이 없이 AT&T의 Korn쉘을 구입할수 있다.

Linux의 판본에서 어떤 쉘들이 사용될수 있는가를 알려면 파일 /etc/shell을 조사하여야 한다. /etc/shell에서 등록된 쉘중의 어느 하나를 변화시키기 위해서는 chsh지령과 쉘의 이름을 입력해야 한다. 영원히 TC쉘에로 변경시키기 위해 chsh지령을 사용한다. 재촉문에서 다음과 같은 형식의 지령을 입력하여야 한다.

```
chsh/bin/tcsh
```

3. 쉘의 력사

초기의 표준UNIX쉘은 1979년 후반기에 V7(AT&T의 7)UNIX에 도입되였으며 후에 이 쉘의 개발자 Stephen Bourne의 이름을 붙였다. 프로그람작성언어로서의 Bourne쉘은 알골(Algol)언어에 기초하고 있으며 초기에 체계관리과제들을 자동적으로 조종하기 위해 사용되였다. 이 쉘은 그의 단순성과 속도로 하여 인기가 있었지만 리력, 별명, 일감조종과 같은 대화형사용을 위한 많은 특징들이 없다. bash쉘인 Bourne쉘에 대하여 보자. 이 쉘은 GNU저작권허가밑에서 자유쏘프트웨어재단(Free Software Foundation)의 Brain Fox에 의해 개발되였으며 일반적인 Linux조작체계에서는 기정쉘이다. 이 쉘은 IEEE POSIX P1003.2/ISO 9945.2쉘과 표준도구에 부합되도록 작성되였다. Bash 역시 최초의 Bourne쉘에 없는 몇개의 새로운 특성(대화동작과 프로그람작성준위에서)들을 제공한다(지금도 Bourne쉘스크립트들을 변화시키지 않고 실행한다.). 이 쉘은 또한 C쉘과 Korn쉘의 가장 유용한 특징들을 통합하고 있는데 크기가 크다. Bourne쉘에 비해 개선된 특성은 지령행리력과 편집기능, 등록부탄창, 일감조종, 함수, 별명, 배렬, 옹근수산수연산(2부터 64까지의 임의의 밑수에서)이며 확장된 메타문자, 차림표를 작성하기 위한 선택순환, let지령 등과 같은 Korn쉘의 특성들이다.

1970년대 후반기에 버클리(Berkeley)에 있는 캘리포니아종합대학에서 개발된 C쉘은 2BSD UNIX의 한 부분으로 계렬화되였다. Bill Joy에 의해 초기에 작성된 쉘은 표준 Bourne쉘에서 제공되지 않은 일련의 추가적인 특징들을 제공하였다. C쉘은 C프로그람작성언어에 기초하며 프로그람작성언어로 사용될 때 류사한 문법을 리용한다. C쉘은 또한 지령행리력, 별명, 일감조종과 같은 대화형리용을 위한 보강된 특성도 제공한다. C쉘은 대형콤퓨터상에서 설계되고 일련의 추가적인 특징들이 보충되였기때문에 소형콤퓨터에서는 속도가 뜨며 Bourne쉘과 비교해 보면 대형콤퓨터상에서도 속도가 뜨다.

TC쉘은 C쉘의 확장판본이다. 일련의 새로운 특징들은 지령행편집(emacs와 vi), 리력목록의 흘리기, 고급한 파일이름, 변수, 지령완료, 맞춤법, 일감일정작성, 자동적인 자물쇠채우기, 등록탈퇴, 리력목록의 시간도장찍기 등이다.

Bourne와 C쉘을 둘다 리용할수 있었기때문에 UNIX사용자는 둘중에 하나를 선택하여야 했다. 따라서 어느것을 리용하겠는가 하는 혼란이 생겼다. AT&T의 다비드 콘(David Korn)은 1980년대 중엽에 Korn쉘을 작성하였다. 이것은 1986년에 계렬화되였으며 공식적으로는 1988년에 UNIX의 SVR4배포부분으로 되였다. Bourne쉘의 웃준위인 Korn쉘은 UNIX체계에서뿐아니라 OS/2, VMS, DOS에서도 실행된다. 이 쉘은 Bourne쉘과 웃방향호환성을 보장하고 C쉘의 많은 특징들을 보충하였으며 다른 쉘보다 빠르고 효률적이다. Korn쉘은 일련의 수정을 통하여 확장되였다. 비록 1993년 판본이 인기를 모으고 있지만 Korn쉘에서 가장 광범히 사용되는 판본은 1988년 판본이다. Linux사용자들은 공개도메인 Korn(Public Domain Korn)쉘 혹은 간단히 pdksh라고 하는 1988년 다비드 콘(David Korn)쉘의 파생인 Korn쉘의 무료판본을 실행할수 있다는것을 알수 있다. 이 쉘은 무료이고 이식할수 있으며 그와 이름이 류사한 Korn쉘과도 충분한 호환성을 보장하도록 하기 위한 사업과 그것이 POSIX와 호환되도록 하기 위한 사업이 진행중에 있다. Paul Falsted에 의해 작성된 TC쉘특성을 가진 또 다른 Korn쉘의 파생인 Z(zsh)쉘도 리용할수 있으며 일련의 Web싸이트들에서는 무료로 사용할수 있다.

4. 쉘의 사용

쉘의 중요한 기능의 하나는 대화식으로 진행될 때 지령행재촉문에서 입력된 지령들을 해석하는것이다. 쉘은 지령행을 구문해석하고 타브, 공백, 행바꾸기로 이루어 지는 공백으로 구분되는 단어들(일명 통표라고 부른다.)로 분리된다. 만일 단어들이 특정한 메타문자들을 포함하면 쉘은 그 메타문자들을 해석한다. 그리고 쉘은 파일입출력과 배경프로쎄스를 처리한다. 지령행이 처리된 다음 쉘은 지령을 탐색하고 실행을 시작한다.

쉘의 다른 중요한 기능은 사용자의 환경을 전용화하는것인데 이것은 보통 쉘초기화파일에서 진행된다. 이 파일들은 말단건들과 창문특성, 탐색경로, 허가, 재촉문, 말단형태를 정의하는 변수들, 창문, 본문처리프로그람, 프로그람작성언어를 위한 서고와 같은 특정의 응용프로그람들을 위하여 요구되는 변수들을 설정하기 위한 정의들을 포함한다. Korn과 C쉘은 또한 사용자가 파일을 지우거나 부주의로 등록탈퇴하지 못하게 하며 과제가 끝났을 때 사용자에게 그것을 알려 주도록 설정되는 내부변수들과 리력과 별명들의 추가적인 기능을 전용화를 위해 제공한다. 쉘은 해석프로그람작성언어로서 사용할수도 있다. 스크립트라는 쉘프로그람은 파일에 렬거된 지령들로 이루어 진다. 프로그람들은 편집기(직결스크립트작성이 허가되지만)에서 작성된다. 이 프로그람들은 변수대입, 조건검사, 순환명령과 같은 기초적인 프로그람작성구조들이 삽입되는 UNIX지령들로 구성된다. 쉘스크립트들은 번역하지 않아도 된다. 쉘은 스크립트의 매개 행이 건반으로 입력됨에 따라 그 매 행을 해석한다. 쉘이 지령을 해석하기때문에 사용자가 항상 이 지령들이 무엇인가를 리해해야 할 필요가 있다. 가장 일반적인 지령목록에 대해서는 부록 1을 참고하기 바란다.

5. 쉘의 임무

쉘의 임무는 재촉문으로부터 입력되는 임의의 지령들이 정확히 실행되도록 하는데 있다. 이 임무에 포함된 내용들은 다음과 같다.

1. 지령을 읽고 지령행을 구문해석한다.
2. 특수문자들을 평가한다.
3. 파이프, 방향바꾸기, 배경프로쎄스를 설정한다.
4. 신호를 조종한다.
5. 실행을 위해 프로그람을 설정한다.

이 매개 추가선택은 개별적인 쉘에서 구체적으로 설명한다.

제 2 절. 체계기동과 등록가입쉘

체계를 기동시킬 때 첫 프로쎄스를 init라고 부른다. 매개 프로쎄스에는 PID라는 프로쎄스식별번호가 있다. init가 첫 프로쎄스이기때문에 PID값은 1이다. Init프로쎄스는 체계를 초기화한 다음 또 다른 프로쎄스를 시동하여 말단들을 열고 그 말단들과 련관되여 있는 표준입력(stdin), 표준출력(stdout), 표준오유(stderr)를 설정한다. 표준입력은 보통 건반으로 하며 표준출력과 표준오유는 현시장치에 나타난다. 이때 등록가입재촉문상태가 사용자말단에 표시된다.

등록가입이름이 입력된후에 암호를 입력할것을 재촉한다. 그다음 /bin/login프로그람은

passwd파일에서 첫번째 마당을 검사하여 등록가입자가 정확한가를 확인한다. 만일 사용자이름이 있으면 다음단계는 입력된 암호를 암호프로그람에 넣어 그것이 정말 정확한 암호인가를 결정한다. 일단 암호가 확인되면 쉘에 넘겨 지는 작업환경을 정의하는 변수들로 구성된 초기환경을 설정한다. HOME, SHELL, USER, LOGNAME변수들에는 passwd파일의 정보로부터 얻은 값들이 대입된다. HOME변수는 홈(뿌리)등록부를 대입받는다. 즉 SHELL변수는 passwd파일에서 마지막입구점인 등록가입쉘의 이름으로 대입된다. USER와 LOGNAME변수는 등록가입이름으로 대입된다. search path변수는 일반적으로 사용되는 편의프로그람이 규정된 등록부에서 찾을수 있도록 하기 위해 설정된다. login이 완료되면 passwd의 마지막입구점에서 찾은 프로그람을 실행할수 있다. 보통 이 프로그람을 쉘이라고 한다. passwd파일에서 마지막입구점이 /bin/csh이면 C쉘 프로그람이 실행된다. passwd파일에서 마지막입구점이 /bin/sh 혹은 빈(null)값이면 Bourne쉘이 기동한다. 만일 마지막입구점이 /bin/ksh이면 Korn쉘이 실행된다. 이 쉘을 login쉘이라고 한다.

쉘이 기동된 다음 체계관리자에 의해 설정된 임의의 체계적인 초기화파일을 검사한 다음 홈등록부를 검사하며 거기에 쉘의 임의의 특수초기화파일들이 있는가를 검사한다. 이러한 임의의 파일들이 존재하면 그것들이 실행된다. 초기화파일은 사용자환경을 보다 전용화하기 위해 사용된다. 이 파일에서 지령들이 실행되면 재촉문은 현시장치에 나타난다. 쉘은 새로운 지령을 입력하기 위해 대기상태에 놓인다.

1. 지령행구문해석

재촉문에서 지령을 입력할 때 쉘은 입력행을 읽고 통표라는 단어로 행을 분리하면서 지령행을 구문해석한다. 통표(token)는 공백과 타브에 의해 분리되며 지령행은 행바꾸기에 의해 완료된다[1]. 그리고 쉘은 첫 단어가 내부지령인가 디스크상의 실행프로그람인가를 검사해 본다. 만일 내부지령이면 쉘은 내부적으로 지령을 실행한다. 그렇지 않으면 쉘은 프로그람이 어디에 있는가를 알기 위해 경로변수에 기입된 등록부를 탐색한다. 지령을 찾으면 쉘은 새로운 프로쎄스를 파생시키고 프로그람을 실행한다. 쉘은 프로그람의 실행이 완료될 때까지 대기하며 필요하면 탈퇴하는 프로그람의 상태를 알린다.재촉문이 나타나며 전체 프로쎄스가 다시 기동한다.

지령행의 처리순서는 다음과 같다.

1. 리력교환이 진행된다(필요하면).
2. 지령행은 통표 혹은 단어로 나누어 진다.
3. 리력이 갱신된다(필요하면).
4. 인용부호가 처리된다.
5. 별명교환과 함수가 정의된다(필요하면).
6. 방향바꾸기, 배경, 파이프가 설정된다.
7. 변수교환($user, $name 등)이 진행된다.
8. 지령교환(today에 대한 표시는 'date'이다.)이 수행된다.
9. 파일이름교환(cat abc.??, rm*.c 등)이 진행된다.
10. 프로그람실행

[1]. 통표로 행을 분해하는 과정을 어휘해석이라고 한다.

2. 지령의 형태

지령이 실행될 때 그것은 별명, 내부지령, 함수, 디스크상의 실행프로그람이다. 별명은 현존지령에 대한 략자이며 C, TC, Bash, Korn쉘에 적용된다. 함수들은 Bourne, Bash, Korn쉘들에 적용된다. 이 함수들은 개별적인 루틴들로 편성되는 지령들의 모임이다. 별명과 함수들은 쉘의 기억기안에서 결정된다. 내부지령은 쉘에 있는 내부루틴이며 실행프로그람은 디스크상에 존재한다. 쉘은 디스크상에서 실행할수 있는 프로그람을 찾기 위해 경로변수를 사용하며 지령이 실행되기전에 2개의 자식프로쎄스를 파생시킨다.

이 프로쎄스는 일정한 시간을 요구한다. 쉘이 지령을 실행할 준비가 되면 다음과 같은 순서로 지령형태들을 평가한다.[2]

1. 별명
2. 예약어
3. 함수(bash)
4. 내부지령
5. 실행프로그람

실례로 지령이 xyz이면 쉘은 xyz가 별명인가를 확인하기 위해 검사한다. 별명이 아니면 내부지령인가, 함수인가를 검사한다. 별명도, 내부지령도, 함수도 아니면 디스크상에 존재하는 실행형지령이여야 한다. 그때 쉘은 지령에 대한 통보를 탐색하여야 한다.

제 3 절. 프로쎄스와 쉘

프로쎄스는 실행되는 프로그람이며 유일한 PID(process identification)번호에 의해 식별된다. 핵심부는 프로쎄스를 조종하고 관리한다. 한개 프로쎄스는 실행프로그람, 자료와 탄창, 프로그람과 탄창지시기, 등록기 그리고 프로그람이 실행되는데 필요한 정보등으로 구성되여 있다. 쉘은 기동할 때 한개 프로쎄스로 된다. 쉘은 그룹의 PID에 의해 식별되는 한개 프로쎄스그룹에 속한다. 한개 프로쎄스그룹만이 한번에 말단을 조종하며 전경(배경의 앞)에서 실행된다. 등록가입할 때 쉘은 말단조종상태에 있으면서 지령을 입력하기 위해 기다림상태에 있다.

쉘은 다른 프로쎄스를 파생시킬수 있다. 실제로 재촉문이나 쉘스크립트로부터 지령을 입력할 때 쉘은 내부코드나 디스크상에서 지령을 찾은 다음에 그 지령이 실행되게 해야 한다. 이것은 system call(체계호출)이라고 하는 핵심부에 대한 호출에 의해 수행된다. 체계호출은 핵심부봉사에 대한 요구이며 프로쎄스가 체계의 하드웨어에로 접근할수 있는 유일한 방도이다. 프로쎄스를 작성하고 실행하고 완료하기 위한 많은 체계호출들이 있다(쉘은 방향바꾸기와 파이프, 지령바꿔넣기, 사용자지령의 실행 등을 수행할 때 핵심부로부터 다른 봉사들을 제공한다.).

새로운 프로쎄스가 실행되도록 하기 위해 쉘이 리용하는 체계호출은 다음절에서 설명한다. 그림 1-2에서 보여 준다.

---

2. 순서 3과 4는 Bourne쉘과 Korn(88)쉘에서 반대이다. 순서 3은 C쉘과 TC쉘에서 사용할수 없다.

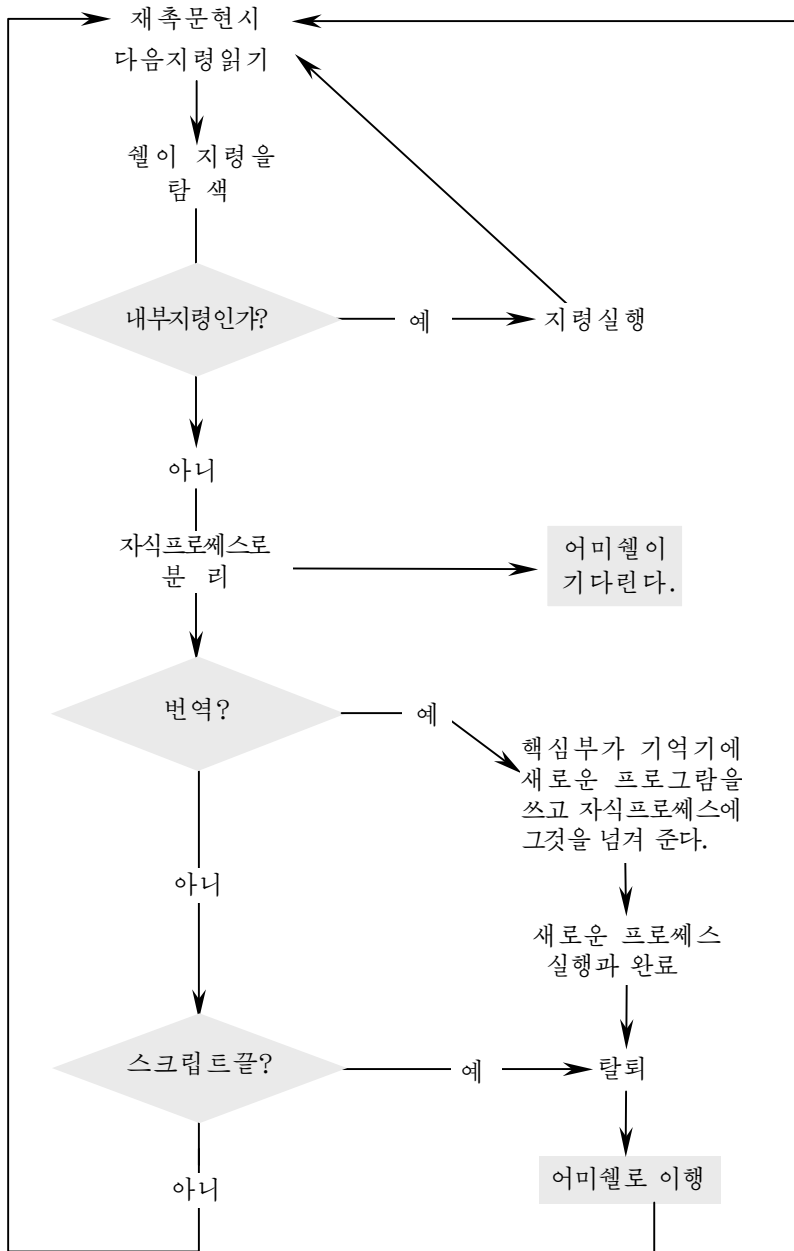



그림 1-2. 쉘과 지령실행

1. 어떤 프로쎄스가 실행되는가

ps지령 많은 추가선택들을 가진 PS지령은 현재 여러가지 형식으로 실행되고 있는 프로그람의 목록을 표시한다. 실례 1-1에 Linux체계에서 사용자에 의해 실행되는 모든 프로쎄스들을 보여 주었다(ps와 그의 추가선택들에 대해서는 부록 1에 보여 주었다.).

실례 1-1

$ ps au *(BSD/Linux ps)* *(use ps –ef for SVR4)*

USER	PID	%CPU	%MEM	SIZE	RSS	TTY	STAT	START	TIME	COMMAND
ellie	456	0.0	1.3	1268	840	1	S	13:23	0:00	-bash
ellie	476	0.0	1.0	1200	648	1	S	13:23	0:00	sh /usr/X11R6/bin/sta
ellie	478	0.0	1.0	2028	676	1	S	13:23	0:00	xinit /home/ellie/.xi
ellie	480	0.0	1.6	1852	1068	1	S	13:23	0:00	fvwm2
ellie	483	0.0	1.3	1660	856	1	S	13:23	0:00	/usr/X11R6/lib/X11/fv
ellie	484	0.0	1.3	1696	868	1	S	13:23	0:00	/usr/X11R6/lib/X11/fv
ellie	487	0.0	2.0	2348	1304	1	S	13:23	0:00	xclock –bg #c0c0c0 –p
ellie	488	0.0	1.1	1620	724	1	S	13:23	0:00	/usr/X11R6/lib/X11/fv
ellie	489	0.0	2.0	2364	1343	1	S	13:23	0:00	xload –nolabel –bg gr
ellie	495	0.0	1.3	1270	848	p0	S	13:24	0:00	-bash
ellie	797	0.0	0.7	852	484	p0	R	14:03	0:00	ps au
root	457	0.0	0.4	724	296	2	S	13:23	0:00	/sbin/minegetty tty2
root	458	0.0	0.4	724	296	3	S	13:23	0:00	/sbin/minegetty tty3
root	459	0.0	0.4	724	296	4	S	13:23	0:00	/sbin/minegetty tty4
root	460	0.0	0.4	724	296	5	S	13:23	0:00	/sbin/minegetty tty5
root	461	0.0	0.4	724	296	6	S	13:23	0:00	/sbin/minegetty tty6
root	479	0.0	4.5	12092	2896	1	S	13:23	0:00	X :0
root	494	0.0	2.5	2768	1632	1	S	13:24	0:00	nxterm –ls -sb -fn

2. 프로쎄스의 창조

fork체계호출 프로쎄스는 UNIX에서 fork체계호출에 의해 작성된다. fork체계호출은 호출하는 프로쎄스를 복사한다. 새로운 프로쎄스를 자식프로쎄스, 작성되는 프로쎄스를 어미프로쎄스라고 부른다. 자식프로쎄스는 fork를 호출하자마자 실행을 시작하며 초기에 두 프로쎄스는 CPU를 공유한다. 자식프로쎄스는 어미프로쎄스환경, 열린 파일, 실제적인 식별과 사용자식별, umask, 현재작업등록부, 신호들을 복사한다.

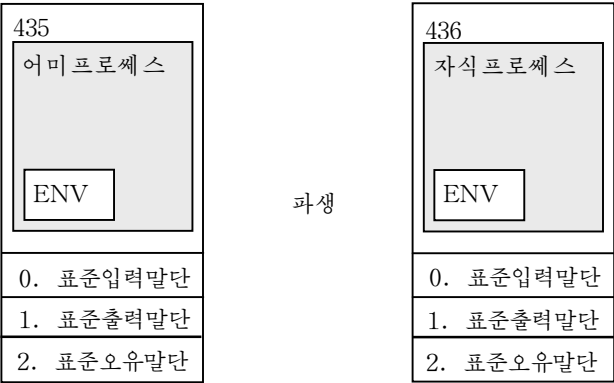



그림 1-3. fork 체계호출

지령을 입력하면 쉘은 지령행을 구문해석하고 첫 단어가 내부지령인가 디스크에 있는 실행지령인가를 결정한다. 만일 지령이 내부지령이면 쉘은 그것을 처리하지만 디스크상에 있으면 복사하기 위해 fork체계호출을 요구한다(그림 1-3). 자식프로쎄스는 방향바꾸기, 파이프, 재촉문, 배경처리를 위한 파일서술자를 설정할뿐만아니라 경로를 탐색하여 지령을 찾는다. 자식쉘이 동작하는 동안 어미쉘은 보통 기다림상태에 있다(아래의 wait를 참고).

wait체계호출 어미쉘은 자식프로쎄스가 방향바꾸기, 파이프, 배경처리를 조종하는것과 같은 동작들을 관리하는 동안 기다리도록 프로그람화된다. wait체계호출은 그의 자식프로쎄스중의 어느 하나가 완료할 때까지 어미프로쎄스가 중지되게 한다. 만일 wait가 성공하면 지우기된 자식프로쎄스의 PID와 탈퇴상태를 되돌려 준다. 어미프로쎄스가 대기상태에 있지 않을 때 자식프로쎄스가 탈퇴하면 자식프로쎄스는 중지상태에 놓이며 어미프로쎄스가 wait를 호출하거나 지우기될 때까지 그 상태를 유지한다.[3] 어미프로쎄스가 자식프로쎄스전에 없어 지면 init프로쎄스는 임의의 중지된 프로쎄스를 취한다.

그다음 wait체계호출은 어미프로쎄스를 대기상태에 놓는데 리용되지 않고 그 프로쎄스가 정확히 완료되였다는것을 담보하는데 사용된다.

exec체계호출 말단에 지령을 준 다음 보통 쉘은 새로운 쉘프로쎄스 즉 자식프로쎄스를 파생시킨다. 이미 언급한바와 같이 자식쉘은 입력한 지령이 실행되도록 한다. 이것을 exec체계를 호출하는 방법으로 수행한다. 사용자지령은 실제로 실행할수 있는 프로그람이다. 쉘은 새로운 프로그람을 찾기 위해 경로를 탐색한다. 만일 찾으면 쉘은 그 변수와 같은 지령이름을 인수로 하여 exec체계호출을 진행한다. 핵심부는 그것을 호출하는 쉘대신에 새로운 프로그람을 기억기에 적재한다. 이때 자식쉘우에 새로운 프로그람이 덧놓인다. 새로운 프로그람은 자식프로쎄스로 되며 실행을 시작한다. 비록 새로운 프로쎄스에 자체의 국부변수가 있지만 모든 환경변수, 열린파일, 신호, 현재작업등록기는 새로운 프로쎄스로 넘어 간다. 이 프로쎄스가 완료되면 탈퇴하며 어미쉘은 대기상태에서 벗어 난다.

exit체계호출 새로운 프로그람은 임의의 시간에 exit호출을 실행하여 완료할수 있다. 자식프로쎄스가 완료되면 신호(sigchild)를 보내고 어미가 그의 탈퇴상태를 접수하기를 대기한다. 탈퇴상태는 0으로부터 255사이의 수값이다. 탈퇴상태값이 령이면 프로그람이 성과적으로 실행되였다는것을 의미하며 령이 아니면 프로그람이 실패하였다는것을 의미한다. 실례로 지령 ls가 지령행에서 입력되였다면 어미쉘은 자식프로쎄스를 파생시키고 대기상태로 들어 간다. 이때 자식쉘은 그 위치에서 ls프로그람을 덧쓰기한다. ls프로그람은 자식프로쎄스대신에 실행되여 모든 환경변수, 열린파일, 사용자정보, 상태정보를 계승한다. 새로운 프로쎄스가 실행을 완료하면 탈퇴하고 어미쉘은 동작상태로 된다. 재촉문은 현시장치에 나타나며 쉘은 다른 지령을 기다린다. 만일 지령이 어떻게 탈퇴했는가를 알려면 매 쉘이 완료된 마지막지령의 탈퇴상태를 포함하는 특수한 내부변수를 가지고 있다는것을 명심해야 한다(이 모든것은 개별적인 쉘을 설명하는 장에서 구체적으로 설명한다.). 프로쎄스작성과 완료의 실례를 그림 1-4에서 보여 주었다.

---

3. 활성이 중지된 프로쎄스를 해제하기 위해서는 체계를 다시 첫넣기하여야 한다.

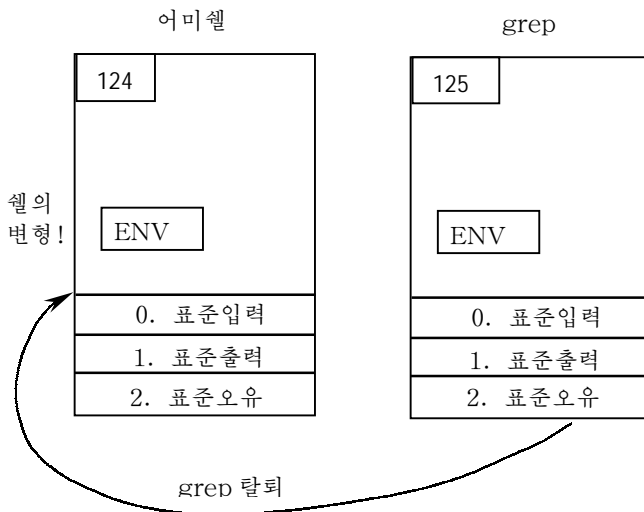
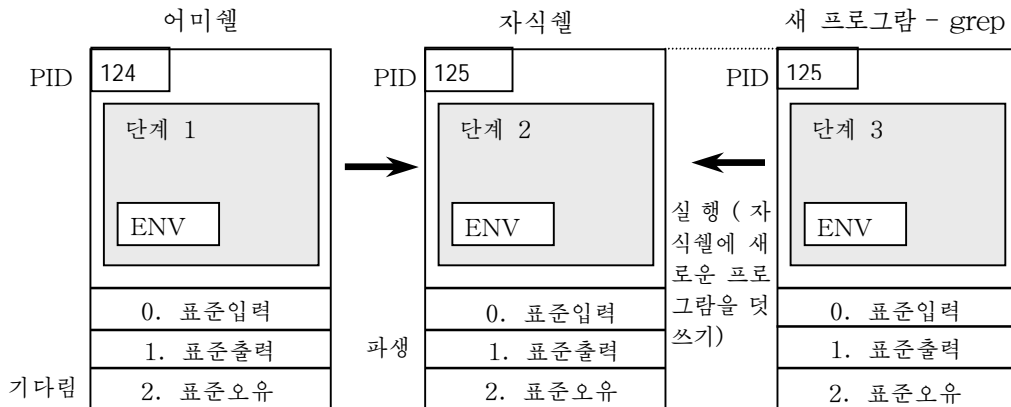



그림 1-4. fork, exec, wait, exit 체계호출

설명

1. 어미쉘은 fork체계호출로 자체의 사본을 만든다. 이 사본을 자식쉘이라고 한다.
2. 자식쉘은 새로운 PID를 가지며 그 어미의 사본이다. 이것은 어미쉘과 함께 CPU를 공유한다.
3. 핵심부는 기억기에 grep프로그람을 적재하고 자식쉘대신에 그것을 실행(exec)한다. grep프로그람은 자식쉘로부터 열린파일과 환경을 계승한다.
4. grep프로그람이 탈퇴하고 핵심부는 지워 지며 어미쉘은 동작준비상태로 된다.

실례 1-2

(C Shell)

1. % **cp filex filey**

```
% echo $status
0

2.  % cp xyz
    Usage: cp [-ip] fl f2; or. cp [-ipr] fl ... fn d2
   % echo $status

   (Bourne and Korn. Shells)
3.  $   cp filex filey

   $ echo $?
   0

   $ cp xyz
   Usage: cp [-ip] fl f2; or: cp [-ipr] fl..   fn d2
   $ echo $?
   1
```

설명

1. cp(copy)지령은 C쉘지령행재촉문에서 입력된다. 지령이 filey라는 filex의 복사를 만든 다음 프로그람이 탈퇴하며 재촉문이 나타난다. csh status변수는 실행되는 마지막지령의 탈퇴상태를 나타낸다. 만일 상태값이 령이면 cp프로그람은 성과적으로 탈퇴하였다는것을 의미한다. 탈퇴상태값이 령이 아니면 cp프로그람은 실패하였다는것을 의미한다.
2. cp지령을 입력할 때 사용자가 2개의 파일이름 즉 원천과 목적파일을 제공하지 못했다. cp프로그람은 현시장치에 오유통보문을 내보내고 상태를 1로 하여 탈퇴한다. 이 값(번호)은 csh상태변수에 기억된다. 임의의 령 아닌 다른 값은 프로그람이 실패하였다는것을 지적한다.
3. Bourne쉘과 Korn쉘은 첫 두 실례에서 C쉘이 진행한것과 같이 cp지령을 처리한다. 유일한 차이점은 Bourne쉘과 Korn쉘이 ststus변수가 아니라 변수에 탈퇴상태를 기억한다는데 있다.

제 4 절. 환경과 계승

등록(론리)가입할 때 쉘은 기동하면서 일정한 변수, I/O흐름, 쉘을 기동시킨 /bin/login프로그람으로부터 프로쎄스특성을 계승한다. 다음으로 다른 쉘이 등록가입 혹은 어미쉘로부터 파생되면 자식쉘(보조쉘)은 어미쉘로부터 일정한 특성들을 계승하게 된다. 보조쉘(자식쉘)은 배경처리, 지령들의 묶음을 처리하고 또는 스크립트들을 실행하기 위하여 기동될수 있다. 자식쉘은 어미쉘로부터 환경을 계승한다. 환경은 프로쎄스허가(누가 프로쎄스를 소유하는가), 작업등록부, 파일작성마스크, 특수변수, 열린파일 그리고 신호로 구성된다.

1. 소유권

등록가입할 때 쉘에 식별자가 주어 진다. 이것은 실제적인 사용자식별자(UID:user identification), 하나 혹은 몇개의 실제적인 그룹식별자(GID:group identification)

그리고 유효한(effective) 사용자식별자와 유효한 그룹식별자(EUID, EGID)를 가진다. 초기에 EUID와 EGID는 실제UID나 GID와 같다. 이 ID번호는 passwd파일에 있으며 사용자와 그룹을 식별하기 위하여 체계에 의해 사용된다. EUID와 EGID는 파일을 읽기, 쓰기, 실행할 때 프로쎄스가 어떤 허가를 호출하는가를 결정한다. 만일 프로쎄스의 EUID와 파일소유자의 실제UID가 동일하다면 프로쎄스는 그 파일에 대한 소유자의 호출허가를 가진다. 프로쎄스의 EUID와 실제GID가 같으면 프로쎄스에는 소유자의 그룹특권이 있다.

/etc/passwd파일로부터 얻게 되는 UID는 등록가입이름과 관련된 정의옹근수이다. 실제UID는 통파암호렬파일에 있는 세번째 마당이다. 등록가입할 때 등록가입쉘에는 실제UID가 대입되며 등록가입쉘로부터 파생된 모든 프로쎄스들은 그의 허가들을 계승한다. 령의 UID를 가지고 실행하는 임의의 프로쎄스는 뿌리(주사용자)에 속하는 뿌리우선권을 가진다. 실지그룹식별자 GID는 그룹을 등록가입이름과 련관시킨다. 그것은 통파암호렬파일의 네번째 마당에 있다.

EUID와 EGID는 서로 다른 소유자에게 대입된 번호로 변경될수 있다. 또 다른 소유자에로 EUID(혹은 EGID[4])를 변경시키는 방법에 의해 그밖의 다른 소유자에게 속하는 프로쎄스의 소유자로 될수 있다. 다른 소유자에게 EUID와 EGID를 변화시키는 프로그람을 setuid와 setgid프로그람이라고 한다. /bin/passwd파일은 사용자에게 뿌리(root)특권을 주는 setuid프로그람의 실례이다. setuid프로그람은 흔히 안전성보장의 견지에서 믿음성이 적다. 쉘은 사용자들이 setuid스크립트를 작성하도록 하며 쉘 그자체도 setuid프로그람으로 될수 있다.

2. 파일창조마스크

파일이 작성될 때 기정적인 허가들의 모임이 주어 진다. 이 허가는 파일을 작성하는 프로그람에 의해 결정된다. 자식프로쎄스들은 그의 어미프로쎄스로부터 기정값마스크들을 계승한다. 사용자는 재촉문에서 umask지령을 입력하기나 쉘의 초기파일에서 그것을 설정하여 쉘에 대한 마스크를 변경시킬수 있다. umask지령을 리용하여 존재하는 마스크로부터 허가를 지우기한다.

초기에 umask는 000이고 등록부에 기정값으로 777(rwxrwxrwx)허가를 주고 파일에 666(rw-rw-rw)허가를 준다. 대부분의 체계에서 umask는 /bin/login프로그람이나 /etc/profile초기파일에 의해 022의 값이 대입된다.

다음과 같이 기정설정값으로부터 umask값을 덜어서 등록부와 파일허가를 얻는다.

777(등록부)	666(파일)
-022(umask 값)	-022(umask 값)
…	…
755	644

결과: drwxr-xr-x   -rw-r—r—

umask가 설정된후 프로쎄스에 의해 작성된 모든 등록부와 파일들은 새로운 기정값에 대한 허가를 대입 받는다. 이 실례에서 등록부들에는 읽기, 쓰기, 실행허가가 주

---

4. setgid허가는 그 사용이 체계에 의존한다. 일부 체계에서 등록부의 setgid는 그 등록부에서 작성된 파일들을 그 등록부에 소유된 같은 그룹에 소속되게 한다. 한편 EGID프로쎄스는 파일을 사용할수 있는 그룹을 결정한다.

어 지며 그룹에 대해서는 읽기와 실행 그리고 나머지것들에 대해서는 읽기와 실행허가가 주어 진다. 작성된 모든 파일들에는 소유자에 대해서 읽기와 쓰기허가가 대입되며 그룹과 기타에 대해서는 읽기허가가 대입된다. 개별적인 허가와 등록부들에 대한 허가를 변경시키기 위해 chmod지령이 사용된다.

3. chmod에 의한 허가변경

모든 UNIX파일에 대해 한개 소유자가 있다. 유일한 소유자 혹은 주사용자(superuser)는 chmod지령을 실행하여 파일 혹은 등록부에 대한 허가를 변경시킬수 있다. 다음의 실례는 허가방식을 설명하였다. 그룹은 몇개의 성원들을 가질수 있고 파일에 대한 그룹허가를 변경시켜 그 그룹이 특권을 행사할수 있게 한다.

chown지령은 파일과 등록부에 대한 소유자와 그룹을 변경시킨다. 소유자 혹은 주사용자(superuser)만이 그것을 호출할수 있다. UNIX의 BSD판본에서는 주사용자 즉 뿌리(root)만이 소유권을 변경시킬수 있다. 파일을 누가 읽고 쓰고 실행할수 있는가를 조종하기 위해 매 UNIX파일은 그것과 관련된 허가모임을 가지고 있다. 9개의 전체 비트들은 파일에 대한 허가들로 구성된다. 여기서 첫 3개 bit모임은 파일소유자의 허가, 두번째 모임은 그룹의 허가, 마지막모임은 그밖의 허가를 조종한다. 허가는 파일의 inode의 mode마당에 기억된다.

chmod지령은 파일과 등록부에 대한 허가를 변경시킨다. 사용자는 파일과 등록부[5]의 허가를 변경시키기 위해 파일을 가지고 있어야 한다. 표 1-1은 허가변경을 위해 사용되는 8개의 가능한 수들의 조합을 보여 주었다.

표 1-1. 허가방식

10진수	8진수	허가
0	000	-
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-w
6	110	rw-
7	111	rwx

실례 1-3

```
1. $ chmod 755 file
   $ ls -1 file
   -rwxr-xr-x 1 ellie 0    Mar   7 12:52 file
2. $ chmod g+w file
   $ Is -1 file
```

5. 호출자의 EUID는 파일자체 UID와 일치되던가 혹은 사용자가 주사용자여야 한다.

-rwxrwxr-x 1 ellie 0 Mar 7 12:54 file

3. $ **chmod go-rx file**
   $ **Is -1 file**
   -rwx-w---- 1 ellie0 Mar 7 12:56 file
4. $ **chmod a=r file**
   $ **Is -1 file**
   -r--r--r-- 1 ellie 0 Mar 7 12:59 file

설명

1. 첫 인수는 8진수값 755이다. 이것은 사용자에 대해서는 rwx로, 그룹에 대해서는 r와 x로, 파일에 대해서는 다른 값으로 설정한다.
2. chmod의 기호형식에서 쓰기허가가 그룹에 추가된다.
3. chmod의 기호형식에서 읽기와 실행허가는 그룹과 기타에서 제외된다.
4. chmod의 기호형식에서는 모든것에 읽기허가만을 준다. 기호 =는 모든 허가를 새로운 값으로 재설정하도록 한다.

실례 1-4

(The Command Line)

1. $ **chown steve filex**
2. $ ls –1

(The Output)
-rwxrwxr-x 1 ***steve*** *groups 170 Jul 28:20 filex*

설명

1. filex의 소유권은 steve로 변경된다.
2. ls-l지령은 3렬에서 소유자 steve를 표시한다.

4. chown 지령을 사용한 소유관계변경

작업등록부 등록가입할 때 파일체계안에서 home등록부라고 하는 작업등록부가 설정된다. 작업등록부는 이 쉘로부터 파생된 프로쎄스에 의해 계승된다.

이 쉘의 임의의 자식프로쎄스는 자체의 작업등록부를 변경시킬수 있으며 그 변경은 어미쉘에 영향을 주지 않는다.

작업등록부를 변경시키는데 사용되는 cd지령은 쉘내부지령이다.

매개 쉘은 cd의 복사를 진행한다. 내부지령은 쉘의 코드부분처럼 쉘에 의해 직접 실행된다. 즉 쉘은 내부지령을 실행할 때 fork와 exec체계호출을 하지 않는다. 다른 쉘(스크립트)이 어미쉘로부터 파생되고 cd지령이 자식쉘에서 설정되면 등록부는 자식쉘에서 변경된다. 자식쉘이 탈퇴할 때 어미쉘은 자식쉘이 기동하기전에 있었던 같은 등록부에 있게 된다.

실례 1-5

1. % **cd /**
2. % **pwd**
   /
3. % **sh**
4. $**cd/home**
5. $ **pwd**
   /home
6. $ **exit**
7. % **pwd**
   /
   %

설명

1. 재촉문은 C쉘의 재촉문이다. cd지령은 등록부를 /으로 변경시킨다. cd지령은 쉘에 내부코드에 내장되여 있다.
2. pwd지령은 어미작업등록부 즉 /문자를 표시한다.
3. Bourne쉘이 기동한다.
4. cd지령은 등록부들을 /home으로 변경시킨다.
5. pwd지령은 현재작업등록부인 /home을 표시한다.
6. Bourne쉘이 탈퇴하여 C쉘로 되돌아 간다.
7. C쉘에서 현재작업등록부는 여전히 /이다. 매 쉘은 자체의 cd사본을 가진다.

변수 쉘은 두 종류의 변수들 즉 국부변수와 환경변수들을 정의할수 있다. 변수에는 쉘을 전용화하기 위해 사용되는 정보와 다른 프로쎄스들이 정확히 동작하기 위하여 요구되는 정보를 포함한다. 국부변수들은 그것들이 작성된 쉘에서만 쓰며 그 쉘로부터 파생되는 임의의 프로쎄스들에 넘어 가지 않는다. 한편 환경변수들은 어미프로쎄스로부터 자식프로쎄스로, 자식프로쎄스로부터 손자프로쎄스 등으로 모두 넘겨 진다. 일부 환경변수들은 등록가입쉘에 의해 /bin/login 프로그람으로부터 계승된다. 다른 변수들은 사용자초기파일, 스크립트 또는 지령행에서 작성된다. 환경변수가 자식쉘에서 설정되면 어미쉘로 다시 넘겨 지지 않는다.

파일서술자 파일, 파이프 그리고 소케트들을 비롯한 모든 입출력들은 핵심부에 의해 파일서술자라고 하는 기구를 통하여 처리된다. 파일서술자는 핵심부에 의해 유지되는 파일서술자표에 대한 첨수로서 부호 없는 옹근수인데 핵심부는 그것을 사용하여 열린 파일들과 I/O흐름을 참조한다. 모든 공정은 어미쉘로부터 자체의 파일서술자표를 계승한다. 첫 3개의 파일서술자 0, 1, 2는 말단에 대입된다. 파일서술자표 0은 표준입력(stdin)이며 1은 표준출력(stdout) 그리고 2는 표준오유(stderr)이다. 파일을 열 때 다음에 사용할수 있는 서술자는 3이며 이것은 새로운 파일에 대입된다. 모든 유효한 파일서술자들이 다 사용되고 있으면 새로운 파일은 열수 없다. [6]

---

6. 내부지령인 limit 와 ulimit 를 보시오.

방향바꾸기(redirection) 파일서술자가 말단이 아닌 다른것으로 대입될 때 그것을 I/O방향바꾸기(redirection)라고 한다. 쉘은 표준출력파일서술자1(말단)을 닫고 그 서술자를 파일에 대입하여 파일에로 출력을 방향바꾸기한다(그림 1-5). 표준입력을 방향바꾸기할 때 쉘은 파일서술자 0(말단)을 닫고 파일에 그 서술자를 파일에 대입한다(그림 1-6).

Bourne쉘과 Korn쉘들은 파일을 파일서술자 2에 대입시켜 오유를 처리한다(그림 1-7). 이와 반면에 C쉘은 같은 기능을 수행하려면 더 복잡한 공정을 거쳐 수행된다(그림 1-8).

실례 1-6

```
1.  % who > file
2.  % cat filel file2 » file3
3.  % mail tom < file
4.  % find / -name file -print 2> errors
5.  % ( find / -name file -print   >   /dev/tty)   >& errors
```

설명

1. who지령의 출력은 말단으로부터 file로 방향바꾸기된다(모든 쉘은 이런 식으로 출력을 방향바꾸기한다.).
2. cat지령(file1과 file2를 련결한다.)의 출력은 file3에 추가된다(모든 쉘은 이런 식으로 출력을 방향바꾸기하여 추가한다.).
3. file의 입력은 mail프로그람에로 방향바꾸기된다. 즉 사용자 tom에 file의 내용이 보내진다(모든 쉘들은 이런 식으로 입력을 방향바꾸기한다.).
4. find지령의 임의의 오유는 errors에 방향바꾸기된다. 출력은 말단으로 전송된다(Bourne쉘과 Korn쉘은 이런 식으로 오유들을 방향바꾸기한다.).
5. find지령의 임의의 오유는 errors에로 방향바꾸기된다. 출력이 말단에 전송된다(C쉘은 이 방법으로 오유들을 방향바꾸기한다.).

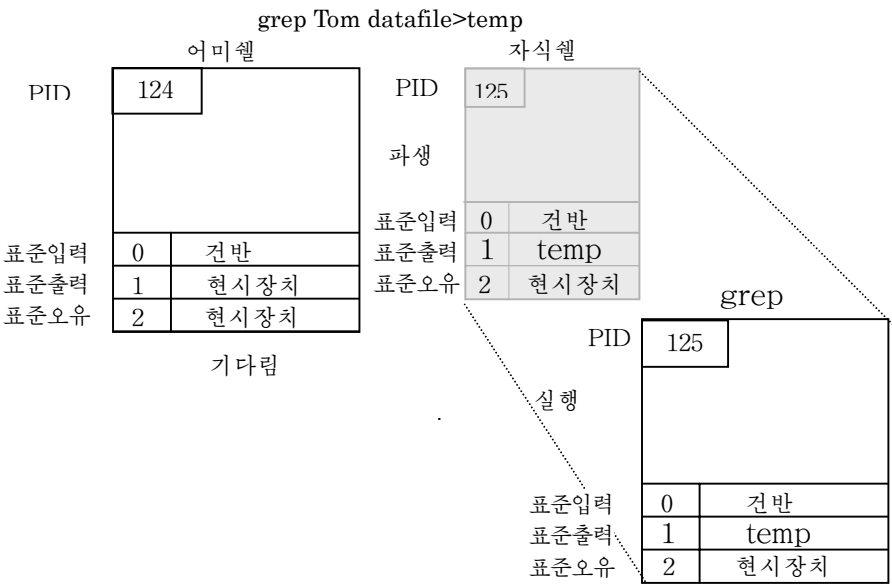



그림 1-5. 표준출력의 방향바꾸기

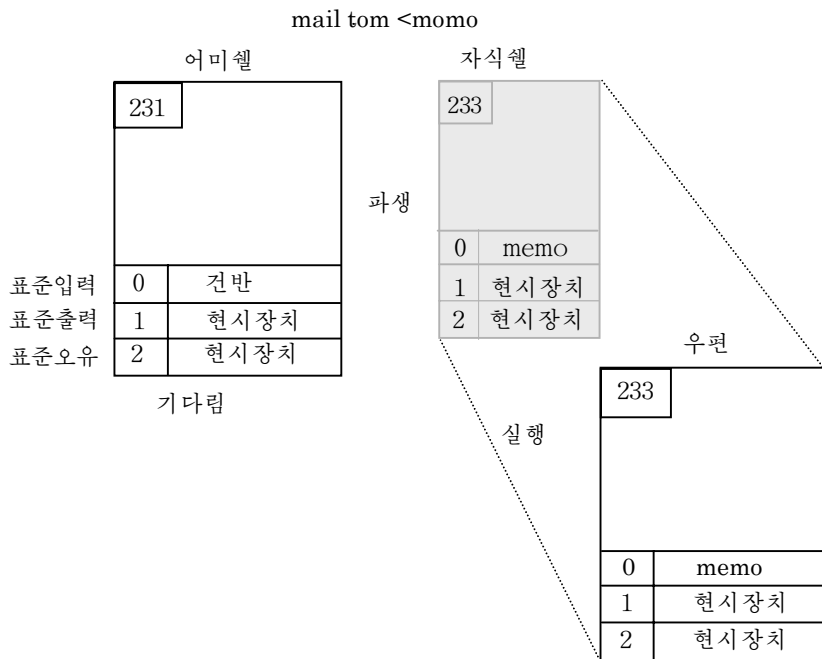



그림 1-6. 표준입력의 방향바꾸기

Find/ -name filex 2>errors

어미쉘

자식쉘

233

235

파 생

0 건반

1 현시장치

2 오유

표준입력 0 건반

표준출력 1 현시장치

표준오유 2 현시장치

기다림

탐색

235

실행

0 건반

1 현시장치

2 오유

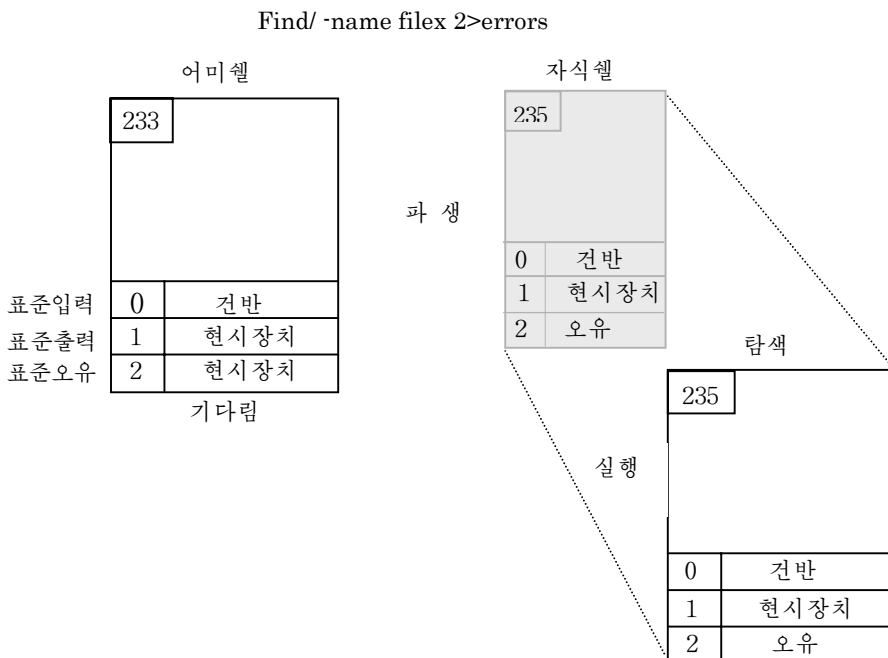



그림 1-7. 표준오유의 방향바꾸기(Bourne 와 Korn 쉘)

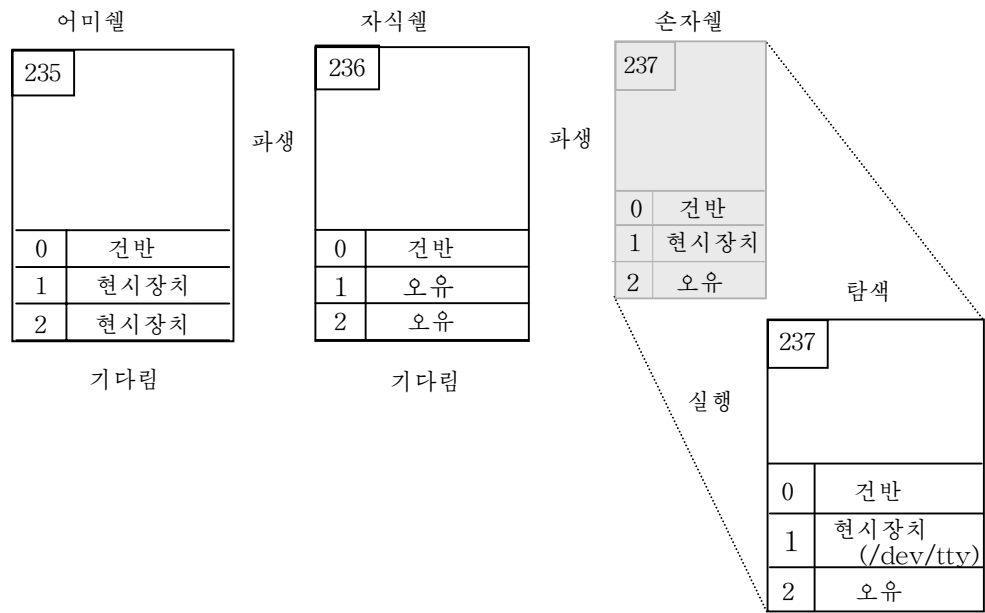



그림 1-8. 표준오유의 방향바꾸기(C 쉘)

파이프 파이프는 한개 지령의 출력이 다른 지령의 입력에로 전송되게 한다. 쉘은 파일서술자를 닫거나 여는 방법으로 파이프를 진행한다. 그러나 쉘은 서술자들을 파일에 대입하지 않고 pipe체계호출을 리용하여 작성된 파이프서술자에 대입한다. 어미쉘이 파이프파일서술자를 작성한 다음 파이프흐름에 있는 매 지령에 대하여 자식프로쎄스를 파생시킨다. 매개 프로쎄스가 파이프서술자를 조종하게 하여 한 프로쎄스는 파이프에 쓰기하고 다른것은 그로부터 읽어 들인다. 파이프는 다만 두 프로쎄스가 자료를 공유하는 핵심부완충기이므로 중간림시파일이 필요 없다. 서술자가 설정된 다음지령은 동시에 실행(exec)된다. 한개 지령의 출력이 완충기에 전송되며 완충기가 충만되거나 지령이 완료될 때 파이프의 오른쪽 지령은 완충기로부터 읽는다. 핵심부는 한개 프로쎄스가 다른것이 완충기로부터 읽거나 쓰는 동안 대기하도록 동기를 맞춘다.

pipe지령의 문법은

```
who | wc
```

이다.

쉘은 who지령의 출력을 입력으로서 wc지령에 보낸다. 이것은 pipe체계호출에 의해 실현된다. 어미쉘은 2개의 파이프서술자 즉 파이프로부터 읽기와 거기로 써넣기 위한 서술자들을 작성하는 pipe체계호출을 진행한다. 파이프서술자와 련관된 파일들은 림시로 자료를 기억하는데 리용되는 핵심부조종 I/O완충기이므로 림시파일을 만들지 않아도 된다.

그림 1-9에서 1-13까지는 파이프의 실행단계를 보여 준다.

(1) 어미쉘은 pipe체계호출을 진행한다. 두 파일서술자 즉 파이프로부터 읽기 위한 서술자와 파이프에 써넣기 위한 서술자는 되돌려 진다. 대입되는 파일서술자들은 파일서술자(fd)표에서 다음번 리용가능한 서술자들인

fd3과 fd4이다(그림 1-9).

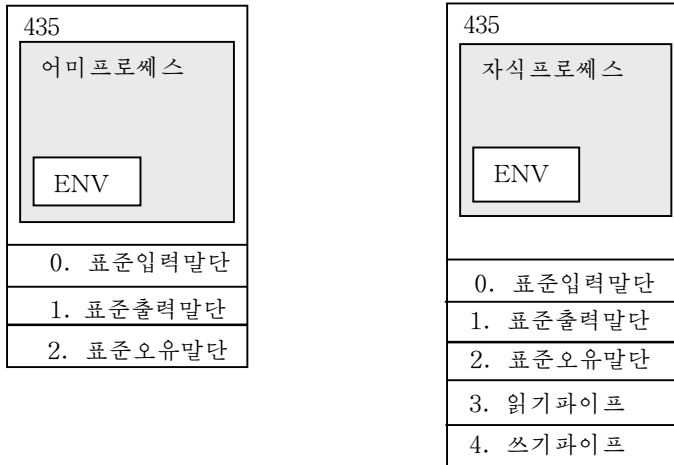



그림 1-9. 어미프로쎄스는 파이프를 설정하기 위한 pipe체계호출을 진행한다.

(2) 매개 지령 who와 wc에 대해 어미프로쎄스는 자식프로쎄스를 파생시킨다. 두 자식프로쎄스는 어미프로쎄스의 열린파일서술자들의 사본을 가진다(그림 1-10).

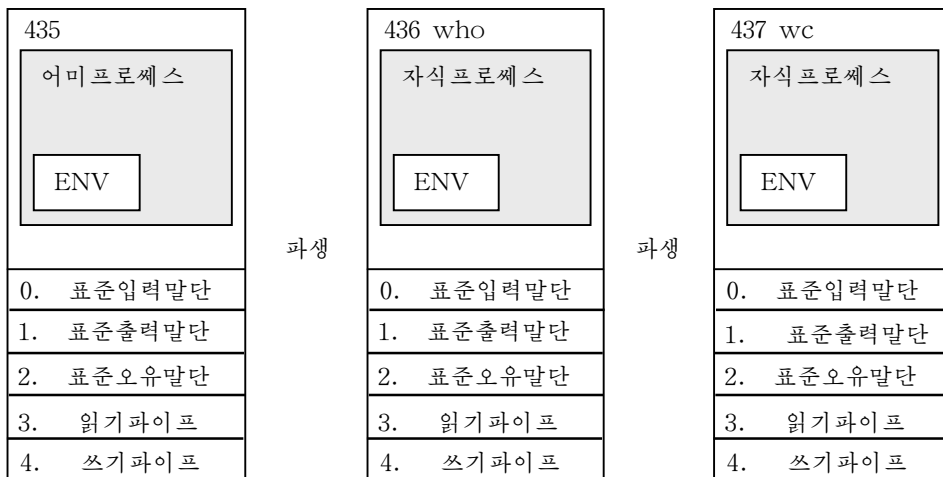



그림 1-10. 어미프로쎄스는 2개의 자식프로쎄스를 파생시키는데 그중에서 하나는 흐름선의 매 지령을 위한것이다.

(3) 첫 자식프로쎄스는 자기의 표준출력을 닫는다. 그리고 파이프에 대한 쓰기와 련관된 파일서술자 4를 복사한다(dup체계호출). dup체계호출은 fd4를 복사하여 그 사본을 표에서 리용가능성이 제일 적은 서술자 fd1에 대입한다. 이때 자식프로쎄스는 fd3이 필요 없으므로 그것을 닫는다. 이 자식프로쎄스는 표준출력 output가 파이프로 나갈것을 요구한다(그림 1-11).

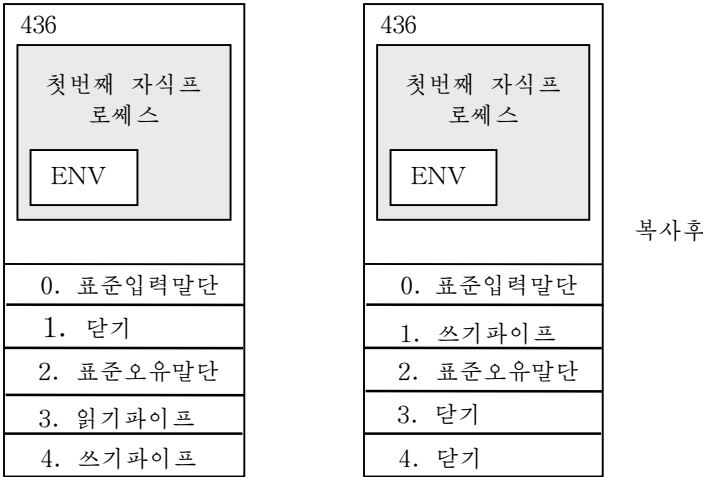



그림 1-11. 첫 자식프로세스는 파이프에 쓰기 위해 준비된다.

(4) 자식프로세스 2는 자기의 표준입력을 닫는다. 그리고 파이프로부터 읽기와 련관된 fd3을 복사한다. dup를 사용하여 fd3을 복사하고 그것을 사용가능성이 제일 적은 서술자에 대입한다. fd0이 닫겨 졌으므로 그것이 사용가능성이 제일 적은 서술자이다. dup는 fd3을 닫는다. 자식프로세스는 fd4를 닫는다. 이 표준입력(input)은 파이프로부터 들어 온다(그림 1-12).

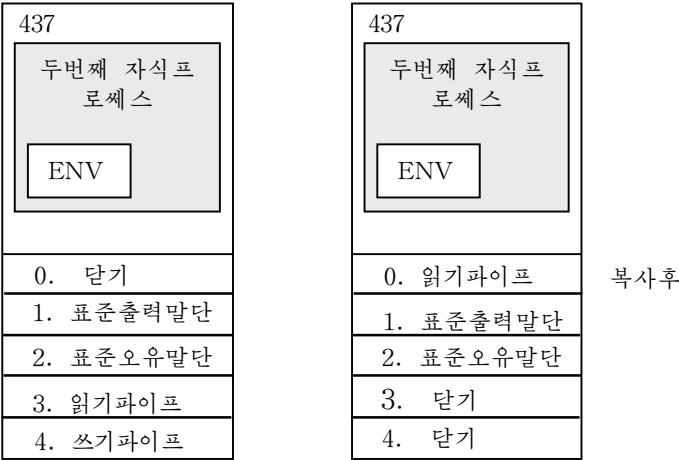



그림 1-12. 두번째 자식프로세스는 파이프로부터 입력을 읽기 위해 준비된다.

(5) who지령은 자식프로쎄스 1대신에 실행되며 wc지령은 자식프로쎄스 2대신에 실행된다. who지령의 출력은 파이프로 나가며 wc지령에 의해 파이프의 다른 끌으로부터 읽어 진다(그림 1-13).

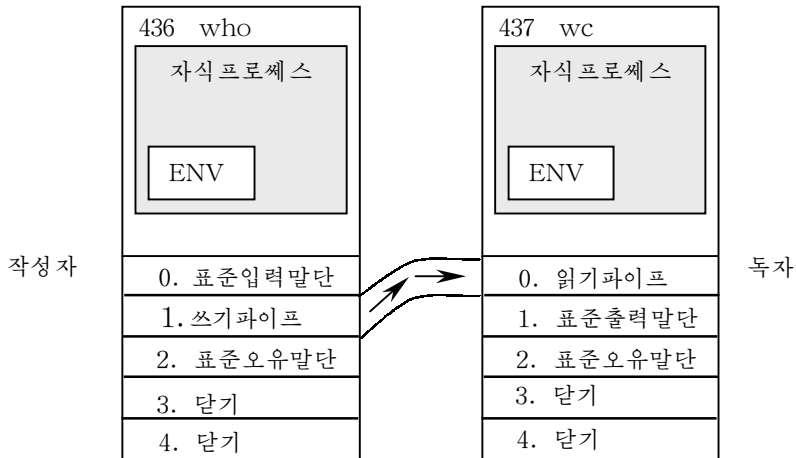



그림 1-13. WHO의 출력은 WC의 입력에 보내진다.

5. 쉘과 신호

신호는 프로쎄스에 통보문을 보내며 보통 구문오유, 모선오유 혹은 전원오유와 같은 일련의 예견하지 못한 사건이 발생할 때 그 프로쎄스를 끝낸다. Break, Delete, Quit 혹은 Stop건을 눌러 프로쎄스에 신호를 보낼수 있으며 그 말단을 공유하는 모든 프로쎄스는 전송된 신호에 의해 영향을 받는다. 사용자는 kill지령으로 프로쎄스를 없앨수 있다. 대부분의 신호들은 기정값에 의해 프로그람을 끝낸다. 쉘은 프로그람으로 들어오는 신호들을 무시하거나 특정의 신호가 도착할 때 어떤 동작을 하게 함으로써 신호들을 조종하게 한다. C쉘은 ^C(Control-C)를 조종하는데서 제한을 받는다.

제 5 절. 스크립트로부터 지령의 실행

쉘이 프로그람언어처럼 사용될 때 지령과 쉘조종구조는 편집기에서 입력되고 스크립트라고 하는 파일에 기억된다. 파일행은 쉘에 의해 한번에 하나씩 읽어 지고 실행된다. 이 프로그람은 해석은 되나 번역되지는 않는다. 번역되는 프로그람은 실행되기전에 기계어로 변환된다. 그러므로 쉘프로그람은 2진실행형프로그람들보다는 속도가 뜨나 작성하기가 쉬우므로 주로 과제를 자동조종하는데 사용된다. 쉘프로그람을 지령행에서 대화형으로 작성할수 있으므로 간단한 과제작성에서는 매우 빠른 방법이다. 그러나 보다 복잡한 스크립트작성에 대해서는 편집기에서 스크립트를 작성하는것이 더 쉽다(훌륭한 타자수가 아니라면). 아래에서 설명하게 되는 스크립트는 임의의 쉘로 실행하여도 동일한 결과를 출력할수 있다. 그림 1-14는 doit라고 하는 스크립트의 작성과 그것이 이미 존재하는 UNIX프로그람 /utilities/commands와 어떻게 맞물리는가를 설명하고 있다.

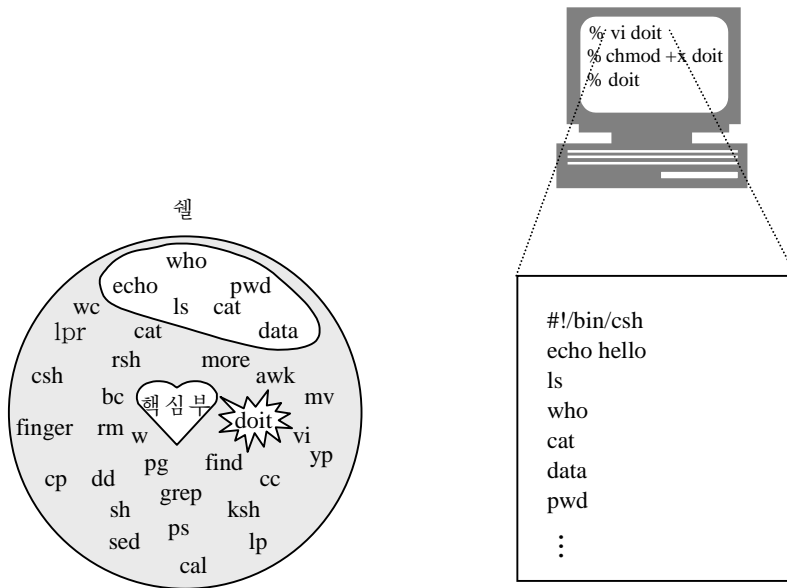



그림 1-14. 일반 쉘스크립트만들기

설명

1. 익숙된 편집기에 넘어 가서 UNIX지령묶음을 한 행에 하나씩 건반으로 입력한다. 첫 행에 있는 #!뒤에 쉘의 경로이름을 붙여 어느 쉘을 요구하는가를 지적한다. 이 프로그람은 C쉘에 의해 실행되며 doit라고 이름을 달았다.
2. 파일을 기억하고 그것을 실행할수 있도록 실행허가를 설정한다.
3. 임의의 다른 UNIX지령들을 실행하는것과 같이 프로그람을 실행한다.

1. 표본스크립트: 3개 쉘의 비교

얼핏 보면 다음의 3개 프로그람들은 아주 비슷한것처럼 보이는데 실지로 류사한 프로그람들이다. 이 프로그람들은 모두 동일한 기능을 수행한다. 기본차이점은 문법에 있다. 일정한 기간 이 3개의 쉘을 리용해 보면 그 차이점을 빨리 알수 있으며 어느 쉘이 자기에게 제일 적합한가에 대한 견해를 세울수 있다. C쉘, Bourne쉘, Korn쉘들사이의 구체적인 차이점을 비교한것을 부록 2에 보여 준다.

다음의 스크립트들은 사용자의 목록에 전자우편통보문을 보내여 그들 모두를 만찬회(야회)에 초청한다. 만찬회(야회)의 장소와 시간은 변수에 설정된다. 초청할 사람들은 guests라는 파일로부터 선택된다. 식료품목록은 단어목록에 기억하며 매 사람들이 목록에 있는 식료품들중의 하나를 가지고 오도록 요구한다. 만일 식료품항목보다 더 많은 사람들이 있으면 목록은 재설정되여 매 사용자가 서로 다른 식료품을 가져 오도록 요구한다. 초청 받지 않은 사람은 사용자 뿌리뿐이다.

2. C 쉘스크립트

실례 1-7

```
1.  #!/bin/csh –f
2.  #   The Party Program- -Invitations to friends from the "guest" file
3.  set guestfile = -/shell/guests
4.  if ( ! –e   "$guestfile" ) then
        echo "$guestfile:t non-existent"
        exit 1
    endif
5.  setenv PLACE "Sarotini's"
    @ Time = 'date +%H' + 1
    set food = ( cheese crackers shrimp drinks "hot dogs" sandwiches )
6.  foreach person ( 'cat $guestfile' )
        if ( $person =~ root .). continue
7.  mail –v -s "Party" $person « FINIS     # Start of here document
    Hi ${person}! Please join me at $PLACE for a party!
    Meet me at $Time o'clock.
    I'll bring the ice cream. Would you please bring $food[l] and   anything else
    you would like to eat? Let me know if you can't   make it. Hope to see you
    soon.
        Your pal,
        Ellie@'hostname'            # or 'uname -n '
   FINISH
8.     shift food ,
       if ( $#food -= 0 .). then
           set food = ( cheese crackers shrimp drinks "hot dogs"
                        sandwiches .).
       endif
9.  end
    echo "Bye..."
```

설명

1. 이 행은 C쉘스크립트가 실행되고 있다는것을 핵심부가 알도록 한다. –f추가선택은 고속기동을 위해 사용한다. 이 행은《.cshrc파일을 실행하지 마시오.》를 알린다. 여기서 .cshrc파일은 새로운 csh프로그람이 시작될 때마다 자동적으로 실행되는 초기화파일이다.
2. 이 행은 설명문이다. 쉘은 이 행을 무시하나 스크립트가 무엇을 수행하고 있는가를 리해하려고 하는 사람들에게는 중요하다.

3. 변수 guestfile은 guests라고 하는 파일의 완전한 경로이름으로 설정된다.
4. 이 행의 의미는 다음과 같다. 만일 파일 guests가 존재하지 않으면 “guests non-existent”를 현시장치에 표시하고 탈퇴상태 1로서 스크립트에서 빠져나와 프로그람에 어떤 오유가 있다는것을 알려 준다.
5. 설정변수들은 장소, 시간, 가져 오려는 식료품목록에 대한 값을 대입 받는다. PLACE변수는 환경변수이다. Time변수는 국부변수이다. @기호는 C쉘에 내부산수연산을 진행하라고 알려 준다. 즉 date지령으로부터 시간을 취하여 변수 Time에 1을 더한다. Time변수는 C쉘이 예약단어 time과 혼돈되는것을 막기 위해 첫 문자를 대문자 T로 한다.
6. 사용자뿌리를 제외하고 손님목록에 있는 매 사람들을 위해 주어 진 장소와 시간에 만찬회에 사람들을 초청하는 우편통보문을 작성하는데 여기서 그들에게 목록에 있는 식료품들중의 하나를 가져 오도록 요구한다.
7. 우편통보문은 here document라고 하는 파일에 작성된다. 사용자정의단어 FINIS로부터 마지막 FINIS까지의 모든 본문을 mail프로그람에 보낸다. foreach순환은 이름목록을 밀기하여 foreach로부터 예약어 end까지 모든 명령을 실행한다.
8. 통보문이 전송된후 다른 사람이 목록에서 다음식료품을 선택할수 있도록 식료품목록이 이동된다. 만일 식료품항목보다 더 많은 사람들이 있으면 식료품목록은 재설정되여 매 사람이 반드시 한개 식료품항목을 가져 오도록 한다.
9. 이것은 순환고리명령의 끝을 표시한다.

3. Bourne 쉘스크립트

실례 1-8

```
1.  #'/bin/sh
2.  # The Party Program—Invitations to friends from the "guest" file
3.  guestfile=/home/3ody/ellie/shell/guests
4.  if [ ' -£ "$guestfile" ]
    then
       echo "'basename $guestfile' non-existent"
       exit 1
    fi
5.  PLACE='Sarotini's"
    export PLACE
    Time='date +%H'
    Time='expr $Time + 1'
    set cheese crackers shrimp drinks "hot dogs" sandwiches
6.  for person in 'cat $guestfile'

    do
```

```
            if [ $person =~ root ]
            then
                contine
            else
7.              mail -v -s "Party" $person «- FINIS
                Hi $(person.).! Please join me at $PLACE for a party!
                Meet me at $Time o'clock.
                I'll bring the ice cream. Would you please bring $1 and
                anything else you would like to eat? Let me know if you
                can't make it. Hope to see you soon.
                        Your pal,
                        ellie@'hostname'
                FINIS
8.            shift
                if [ $# -eq 0 ]
                then
                    set cheese crackers shrimp drinks "hot dogs" sandwiches
                fi
        fi
9.  done
    echo "Bye..."
```

설명

1. 이 행은 핵심부가 Bourne쉘스크립트를 실행하고 있다는것을 알려 준다.
2. 이 행은 설명문이다. 쉘에서는 무시되나 스크립트가 무엇을 수행하는가를 리해하려고 하는 사람들에게는 중요한 부분이다.
3. 변수 guestfile은 guests 라는 파일의 완전한 경로이름으로 설정된다.
4. 이 행은 다음과 같은 의미를 가진다. 즉 파일 guests가 존재하지 않으면 현시장치에 《guests -non-existent》라는 통보문을 출력하고 스크립트로부터 빠져 나온다.
5. 변수들은 장소와 시간값들을 대입 받는다. 가져 오려는 식료품목록은 set지령으로서 특별한 변수(장소파라메터)에 대입된다.
6. 사용자뿌리를 제외하고 손님목록에 있는 매 사람을 제정된 장소와 시간에만찬회에 초청하고 목록에서 한개 식료품을 가져 오도록 요구하는 우편통보문을 작성한다.
7. 이 행에 대한 설명을 하지 않을 때 우편통보문이 전송된다. 프로그람이 모두 오유수정될 때까지 이 행을 설명하지 않는것은 좋지 않지만 그렇게 하지 않으면 스크립트가 검사될 때마다 전자우편이 같은 사람들에게 보내진다.

here문서와 함께 cat지령을 사용한 다음 명령문은 스크립트가 현시장치에 출력을 전송하는것을 검사하도록 하는데 이것은 7행이 해석되지 않을 때 전자우편을 통하여 보낼수 있다.

8. 통보문이 전송된후 식료품목록은 이동되여 다른 사람이 목록에서 다음식료품을 선택할수 있도록 된다. 식료품보다 더 많은 사람들이 있으면 식료품목록은 재설정되여 매 사람에게 한개의 식료품이 대입되게 한다.
9. 이것은 순환명령문의 끝을 표시한다.

4. Korn 쉘스크립트

실례 1-9

```
1.  #!/bin/ksh
2.  # The Parky Program—Invitations to friends from the "guest" file
3.  guestfile=~/shell/guests
4.  if [[ ! -a "$guestfile" ]]
     then
         print "$(guestfile##*/.). non-existent"
         exit 1
     fi
5.  export PLACE="Sarotini's"
    (( Time=$(date +%H) + 1 .)..).
    set cheese crackers shrimp drinks "hot dogs" sandwiches
6.  for person in $(< $guestfile.).
 do
     if [[ $person = root ]]
     then
                    continue
               else
                        # Start of here document
7.  mail -v -s "Party" $person «- FINIS
          Hi ${person}! Please join me at $PLACE for a party!
     Meet me at $Time o'clock.
     I'll bring the ice cream. Would you please bring $1 and anything else you
     would like to eat? Let me know if you can't make it.
         Hope to see you soon.
```

```
            Your pal,
            ellie@'hostname'
                    FINIS
    8.      shift
                    if (( $# ==   0 ).).).
                    then
                        set cheese crackers shrimp drinks "hot dogs" sandwiches
            fi
            fi
    9.  done
            print "Bye..."
```

설명

1. 이 행은 핵심부가 Korn쉘에서 실행된다는것을 알리는 명령이다.
2. 이 행은 해설이다. 쉘에서는 무시되나 스크립트가 무엇을 수행하는가를 리해하기 위해 애쓰는 사람들에게는 중요하다.
3. 변수 guestfile은 guests라는 파일의 완전한 경로이름으로 설정된다.
4. 이 행은 다음과 같은 의미를 가진다. 만일 guests가 존재하지 않으면 현시장치에 "guests non-existent"를 인쇄하고 스크립트에서 빠져 나온다.
5. 변수들에는 장소와 시간값이 대입된다. 가져 오려는 식료품목록은 set지령에 의해 특별한 변수(위치파라메터)들에 대입된다.
6. 사용자뿌리를 제외하고 손님목록에 있는 매 사람을 제정된 장소와 시간에 만찬회에 초청하고 목록에서 한개 식료품을 가져 오도록 하는 우편통보문을 작성한다.
7. 우편통보문이 전송된다. 통보문의 본체는 here document에 포함되여 있다.
8. 통보문이 전송된후 식료품목록은 다른 사람이 목록에서 다음식료품을 선정할수 있도록 하기 위하여 이동된다. 식료품보다 더 많은 사람들이 있으면 식료품목록은 재설정되여 매 사람들이 식료품을 안전하게 배정 받게 한다.
9. 이것은 순환명령의 끝을 표시한다.

제 2 장. UNIX개발도구칸(toolbox)

리용할수 있는 수많은 UNIX편의프로그람들이 있으며 그들중 많은것은 ls, pwd, who, vi와 같이 일상적으로 많이 쓰이는 지령들이다. 목수가 사용하는 필수적인 도구가 있는것과 같이 프로그람작성자에게도 의의가 있고 유효한 스크립트를 서술하기 위해 필요한 중요한 도구가 있다. 여기서 구체적으로 론의하려는 3개의 중요한 편의프로그람들은 grep, sed, awk이다. 이 프로그람들은 본문, 파이프로부터의 출력 또는 표준입력을 조작하는데 사용할수 있는 가장 중요한 UNIX도구들이다. 사실 sed와 awk는 흔히 그자체가 스크립트작성언어로서 사용된다. grep, sed, awk에 대한 능력을 완전히 파악하려면 정규식과 정규표현메타문자의 사용에 대한 충분한 기초지식을 가져야 한다. 유용한 UNIX편의프로그람의 전체 목록을 부록 1에 보여 주었다.

제 1 절. 정규식

1. 정의와 실례

정규표현메타문자의 개념을 이미 잘 알고 있는 사용자들에게는 이 부분이 무시될수도 있다. 그러나 이 예비자료는 grep, sed, awk를 사용하여 자료를 표시하고 조작하는 여러가지 방법을 리해하는데서 중요하다.

정규식이란 무엇인가? 정규식[1]은 탐색에서 동일한 문자와 일치시키기 위해 사용되는 문자들의 패턴(pattern)이다. 대부분의 프로그람에서 정규식은 빗선문자안에 있다. 실례로 /love/는 빗선문자(/)에 의해 구분되는 정규식이며 패턴 love는 패턴이 탐색되고 있는 행에서 같은 패턴이 발견될 때 일치된다. 정규식을 리용하게 하는것은 그것이 특별한 메타문자에 의해 조종될수 있기때문이다. 만일 정규식에 대한 개념이 없으면 이 전체의 개념을 리해하는데 도움을 줄수 있는 실례를 보면 알수 있다. 친구에게 전자우편통보문을 보내기 위해 vi편집기에서 작업하고 있다고 하자. 그 통보문은 다음과 같다.

```
% vi letter
-----------------------------------------------
Hi tom
I think I failed my anatomy test yesterday.I had a terrible
stomach ache. I ate too many fried green tomatoes.
Anyway, Tom, I need your help. I 'd like to make the test up
tomorrow,but don 't know where to begin studying. Do you
think you could help me? After work,about 7 PM,come to
my place and I 'll treat you to pizza in return for your help.Thanks.
          Your pal,
         Guy @ phantom
~
~
-----------------------------------------------
```

[1] 문자렬 RE 를 포함하는 오유통보문을 받으면 프로그람에서 사용하고 있는 정규표현에 문제가 있다.

톰(Tom)은 검사를 하지 않았고 다비드(David)는 검사를 했다는것을 알았다고 하자. 또한 인사말에서 Tom을 소문자 t로 썼다는것을 발견했다고 하자. 따라서 다음과 같이 전체 교환을 진행하여 모든 tom을 David로 바꾸려고 한다.

```
% vi letter
-----------------------------------------------------------
Hi David
I think I failed my anaDavidy test yesterday.I had a terrible
sDavidachache. I ate too many fried green Davidates.
Anyway, Tom, I need your help. I 'd like to make the test up
Davidorrow,but don 't know where to begin studying. Do you
think you could help me? After work,about 7 PM,come to
my place and I 'll treat you to pizza in return for your help.Thanks.
                              Your pal,
                               guy @ phanDavid
~
~
--> :1,$s/tom/David/g
-----------------------------------------------------------
```

탐색문자렬에서 정규식은 tom이고 교체문자렬은 David이다. vi지령은 《행 1부터 파일의 끝($)까지 매 행에서 모든 tom을 찾아 David로 교체하시오.》와 같다. 이렇게 하면 기대했던 결과를 얻을수 없다. Tom이 아니라 tom이 David로 교체되도록 요구했기때문에 한개의 Tom은 치환되지 않았다. 그러면 어떻게 하겠는가?

정규표현메타문자들은 어떤 교체가 진행되는가를 조종할수 있도록 패턴을 구분할수 있게 하는 특수문자이다. 여기에는 행의 시작이나 끝으로 한개 단어를 고정시키기 위한 메타문자들이 있다. 또한 대문자, 소문자, 수자 등을 찾기 위하여 임의의 문자 혹은 일정한 수의 문자들을 지정하게 하는 메타문자들이 있다. 실례로 이름 tom이나 Tom을 David로 바꾸기 위해 다음의 vi지령을 실행하여야 한다.

```
:1,$s/\ <[Tt]om\ >/David/g
```

이 지령은 《파일의 첫 행으로부터 마지막행(1, $)까지 단어 Tom 혹은 tom을 David로 교체하시오.》라는 의미를 가지며 g기발은 이것을 전반적으로 진행하라는것을 의미한다(즉 동일한 행에 한번이상 나타나면 치환을 진행한다.). 정규표현메타문자에서 단어의 시작과 끝은 \ <과 \ >이며 중괄호의 쌍 [Tt]는 그안에 있는 문자들중 한개를 대신한다(이 경우에 T 혹은 t에 대하여). 모든 UNIX의 패턴일치편의프로그람들이 인식하는 5개의 기본메타문자들과 프로그람에 따라서 서로 다른 메타문자들의 확장된 모임이 있다.

2. 정규표현메타문자

표 2-1에 vi, ex, grep, egrep, awk의 모든 판본들에서 사용될수 있는 정규표현메타문자들을 보여 주었다. 추가적인 메타문자들은 사용할수 있는 매개 편의프로그람에서 설명한다.

표 2-1. 정규표현메타문자

메타문자	기능	실례	무엇을 일치시키는가
^	행시작문자	/^love/	love로 시작하는 모든 행을 일치시킨다.
$	행의 끝문자	/love$/	love로 끝나는 모든 행을 일치시킨다.
.	한개 문자를 일치시킨다.	/l..e/	L과 그뒤에 2개의 문자 그리고 e를 포함하는 행을 일치시킨다.
*	령 혹은 그이상의 선행문자들을 일치시킨다.	/*love/	령 혹은 그이상의 공백과 그뒤에 패턴 love를 가진 행을 일치시킨다.
[]	묶음중의 하나를 일치시킨다.	/[Ll]ove/	Love 나 love를 포함하고 있는 행들을 일치시킨다.
[x-y]	묶음의 범위안에서 문자를 일치시킨다.	/[A-Z]ove/	A~Z사이의 문자에 ove가 뒤에 놓이는 문자들을 일치시킨다.
[^]	묶음밖에서 한개 문자를 일치시킨다.	/[^A-Z]/	love로 시작하는 한개 단어를 포함하고 있는 행들을 일치시킨다(vi와 grep에서 제공한다.).
\	한개 메타문자에서 빠져 나오기(escape) 위해 사용된다.	/love\ >/	love로 끝나는 한개 단어를 포함하는 행들을 일치시킨다(vi와 grep에서 제공한다.).
보충적인 메타문자들은 RE메타문자들을 사용하는 많은 UNIX프로그람에 의해 제공된다.			
\ <	단어시작문자	^<love/	love로 시작하는 한개 단어를 포함하고 있는 행들을 일치시킨다(vi와 grep에서 제공한다.).
\ >	단어의 끝문자	/love\ >	/love로 끝나는 한개 단어를 포함하는 행들을 일치시킨다(vi와 gerp에서 제공한다.).
\ (..\ )	꼬리표는 후에 리용되게 문자를 일치시킨다.	(love\ )able \ 1er/>/	패턴의 제일 왼쪽 부분에서 첫 꼬리표를 시작하여 9개의 꼬리표까지 사용한다. 실례로 패턴 love는 후에 \ 1로서 참조하기 위해 보존된다. 즉 이 실례에서 탐색패턴은 loveable과 그뒤의 lover로 구성되였다(sed, vi, grep에 의해 제공된다.).
x\ {m\ } 혹은x \ {m,\ } 혹은 x \ {m,n\ }	문자 x를 m번 반복, 적어도 m번 반복, 적어도 m번과 n번[ㄱ]보다 많지 않게 반복	o\ {5,10\ }	문자 o를 5개부터 10개까지 포함하는 행을 일치시킨다(vi와 grep에서 제공한다.).

ㄱ. UNIX의 모든 판본이나 모든 패턴일치편의프로그람상에 의존하지는 않는다. 보통 vi와 grep와 함께 동작한다.

vi편집기가 어떻게 동작하는가를 알면 매개 메타문자를 vi탐색문자렬로 표시한다. 다음의 실례에서 문자들을 강조시켜 vi가 탐색에서 찾은것들을 보여 준다.

실례 2-1

(A Simple Regular Expression Search)
% **vi picnic**

---

had a **love**ly time on our little picnic.
Lovers were all around us. It is springtime. Oh
Love, how much I adore you. Do you know
The **extent** of my **love?** Oh, by the way, I think
I lost my g**love**s somewhere out in that field of **love**
c **lover.** Did you see them? I can only hope ***love***
is forever. I live for you. It's hard to get back in the
groove.

~
~
~
/love/

설명

정규식은 love이다. 패턴 love는 그자체와 lovely, gloves, clover와 같은 다른 단어들의 부분으로서 탐색된다.

실례 2-2

(The Beginning-of-Line Anchor (^))
% **vi picnic**

I had a lovely time on our little picnic.
Lovers were all around us. It is springtime. Oh
love, how much I adore you. Do you know
the extent of my love? Oh, by the way, I think
I lost my gloves somewhere out in that field of
clover. Did you see them? I can only hope
love
groove.
~

~

```
~
/^love/
--------------------------------------------------------------------------------------
```

설명

탈자기호(^)를 행의 시작기호라고 부른다. vi는 정규식 love가 행의 시작에서 일치되는 행들만을 찾는다. 즉 love는 행에 있는 첫번째 문자들의 모임이다. 그 앞에는 한개의 공백도 있을수 없다.

실례 2-3

(The End-of-Line Anchor ($)

```
% vi picnic
--------------------------------------------------------------------------------------
I had a. lovely time on our little picnic.
Lovers were all around us. It is springtime. Oh
Love, how much I adore you. Do you know
the extent of my love? Oh, by the way, I think
I lost my gloves somewhere out in that field of
Clover. Did you see them? I can only hope love
is forever. I live for you. It's hard to get back in the
groove.
~
~
~
/love$/
--------------------------------------------------------------------------------------
```

설명

화페기호($)를 행의 끌기호라고 한다. vi는 정규식 love가 행의 끝에서 일치되는 행들만을 탐색한다. 즉 love는 행에서 문자들의 마지막모임이며 그다음에 새 행문자가 놓인다.

실례 2-4

(Any Single Character (. .))

```
% vi picnic
--------------------------------------------------------------------------------------
I had a lovely time on our little picnic
Lovers were all around us. It is springtime. Oh
love, how much I adore you. Do you know
    the extent of my love? Oh, by the way, I think
    I lost my gloves somewhere out in that field of
    clover. Did you see them? I can only hope love
    is forever. I live for you. It's hard to get back in the
    groove.
    ~
```

```
~
~
/l.ve/
--------------------------------------------------------------------------------
```

설명

점(.)은 행바꾸기문자를 제외하고 임의의 한개 문자를 대신한다. vi는 정규식이 l과 그뒤에 오는 임의의 한개 문자 그리고 v와 e로 이루어 지는 행들을 탐색한다. 그것은 love와 live를 탐색한다.

실례 2-5

(zero or More of the Preceding Character (*))

```
% vi picnic
--------------------------------------------------------------------------------
I had a lovely time on our little picnic.
Lovers were all around us. It is springtime. Oh
Love,, how much I adore you. Do you know
the extent of my love? Oh, by the way, I think
I lost my gloves somewhere out in that field of
clover. Did you see them? I can only hope love
is forever. I live for you. It's hard to get back in the
groove.
~
~
/o*ve/
--------------------------------------------------------------------------------
```

설명

별표(*)는 앞에 놓이는 임의의 수의 문자를 대신한다.[2] 별표는 마치도 자기앞에 있는 문자에 붙어 있는것처럼 그 문자만을 조종한다. 이 경우에 별표는 문자 o에 붙는다. 별표는 한개의 o문자만을 대신하거나 패턴에 있는것과 같은 개수의 련속적인 문자 o을 (문자 o이 하나도 없는 경우도 포함) 대신한다. 즉 vi는 ve앞에 빈값 혹은 한개이상의 문자 o가 붙어 있는 문자렬을 탐색하여 love, loooove, lve 등을 찾는다

실례 2-6

(A Set of Characters ([ ]))

```
% vi picnic
--------------------------------------------------------------------------------
I had a lovely time on our little picnic.
Lovers were all around us. It is springtime. Oh
```

2. 쉘통용기호(*)와 이 메타문자를 혼돈하지 말아야 한다. 이것들은 총체적으로 차이난다. 쉘별표는 빈값 혹은 한개이상의 문자를 일치시키며 정규표현별표는 빈값 혹은 한개이상의 앞문자를 일치시킨다.

```
Love, how much I adore you. Do you know
the extent of my love? Oh, by the way, I think
I lost my gloves somewhere out in that field of
Clover. Did you see them? I can only hope love
is forever. I live for you. It's hard to get back in the
groove.
~
~
~
/[Ll]ove/
---------------------------------------------------------------------------------------
```

설명

중괄호는 문자묶음중의 하나를 일치시킨다. vi는 대문자 혹은 소문자 l 다음에 o, v, e가 놓이는 정규식을 탐색한다.

실례 2-7

(A Range of Characters ([– l]))

```
% vi picnic
---------------------------------------------------------------------------
I had a lovely time on our little picnic.
Lovers were all around us. It is springtime. Oh
Love, how much I adore you- Do you know
the extent of my love? Oh, by the way, I think
I lost my gloves somewhere out in that field of
Clover. Did you see them? I can only hope love
is forever. I live for you. It's hard to get back in the
groove.
~
~
~
/ove[a-z]/
---------------------------------------------------------------------------
```

설명

중괄호안에 있는 문자들사이의 횡선[-]은 문자들의 범위내에서 한개 문자를 일치시킨다. vi는 o, v, e 다음에 a와 z사이의 ASCII코드범위에서 임의의 문자를 포함하고 있는 정규식을 탐색한다. 이것은 ASCII코드범위의 문자이므로 범위를 [z-a]로 표현할수 없다.

실례 2-8

(Not One of the Characters in the Set ([^].)).

```
% vi picnic
---------------------------------------------------------------------------------------
```

```
I had a lovely time on our little picnic.
Lovers were all around us. It is springtime. Oh
Love, how much I adore you. Do you know
the extent of my love? Oh, by the way, I think
I lost my gloves somewhere out in that field of
Clover. Did you see them? I can only hope love
is forever. I live for you. It's hard to get back in the
groove.
~
~
~
/ove[^a-zA-z0-9]/
--------------------------------------------------------------------------------------------
```

설명

중괄호안의 기호 ^는 부정메타문자이다. vi는 o, v, e 와 그뒤에 ASCII코드범위에서 a와 z사이, A와 Z사이 그리고 0과 9사이에 놓이지 않는 임의의 문자를 포함하고 있는 정규식을 탐색한다. 실례로 ove 다음에 반점, 점 등이 놓이는 문자를 탐색한다. 왜냐하면 이 문자들은 문자모임에 없기때문이다.

제 2 절. 정규표현메타문자의 조합

지금까지 기본정규표현메타문자들을 설명하였는데 이것들은 보다 복잡한 표현식으로 조합할수 있다. 빗선기호안에 있는 매개 정규식실례들은 탐색문자렬이며 본문파일의 매개 행에 대하여 일치된다.

실례 2-9

Note: The *line numbers are NOT part* of the *text file. The vertical bars mark the left and right margins.*

```
1. | Christian Scott lives here and will put on a Christmas party.|
2. |There are around 30 to 35 people invited.                    |
3. | They are:                                                    |
4. |                                                          Tom |
5. | Dan                                                          |
6. | Rhonda Savage                                                |
7. | Nicky and Kimberly.                                          |
8. | Steve, Suzanne, Ginger and Larry.                            |
```

설명

ㄱ. / ^[A-Z]…$/

이것은 대문자로 시작되고 그뒤로 임의의 두 문자와 행바꾸기문자가 놓이는 모든 행을 검사한다. 5행의 Dan을 탐색한다.

ㄴ. / ^[A-Z][a-z]*3[0-5]/
이것은 대문자로 시작되고 그뒤에 임의의 수의 소문자 혹은 공백, 수자 3 그리고 0과 5사이의 수자가 놓이는 모든 행을 검사한다. 행 2를 탐색한다.
ㄷ. /[a-z]* \ ./
임의의 수의 소문자로 시작되고 그뒤에 점이 놓이는 행을 찾는다. 행 1, 2, 7, 8을 찾는다.
ㄹ. / ^*[A-Z][a-z][a-z]$/
임의의 수의 공백(타브는 공백으로 보지 않는다.)으로 시작되면서 그뒤로 한개 대문자, 2개의 소문자 그리고 행바꾸기문자가 놓이는 행을 탐색한다. 4행에서는 Tom, 5행에서는 Dan을 찾는다.
ㅁ. /^[A-Za-z]*[^,][A-Za-z]*$/
임의의 수의 대문자나 소문자로 시작되면서 그뒤에 반점이 아닌 문자, 임의의 수의 대문자, 소문자와 그리고 행바꾸기문자가 놓이는 행을 찾는다. 행 5를 찾는다.

1. 기타 정규표현메타문자

다음의 메타문자들은 정규식을 사용하는 모든 편의프로그람들에 넘겨 지지는 않지만 vi편집기와 sed, grep의 일련의 판본에서 사용할수 있다. egrep, awk와 함께 사용할수 있는 확장된 메타문자의 모임이 있는데 다음절에서 설명한다.

실례 2-10

(Beginning-of-Word (\ <) and End-of-Word (\ >) Anchors)

```
% vi textfile
------------------------------------------------------------
    Unusual occurrences happened at the fair.
--> Patty won fourth place in the 50 yard dash square and fair.
    Occurrences like this are rare.
    The winning ticket is 55222.
    The ticket I got is 54333 and Dee got 55544.
    Guy fell down while running around the south bend in his last event.
    ~
    ~
    ~
    /\ <fourth\ >/
------------------------------------------------------------
```

설명

매개 행에서 단어 fourth를 찾는다. \ <는 단어의 시작문자, \ >는 단어의 끌문자이다. 단어는 공백, 구두점, 행의 시작에서는 시작점, 행의 끌에서는 끌점 등으로 분리될수 있다.

실례 2-11

```
% vi textfile
------------------------------------------------------------
```

Unusual occurrences happened at the fair.
- -> Patty won ***fourth*** place in the 50 yard dash square and fair.
Occurrences like this are rare.
The winning ticket is 55222.
The ticket I got is 54333 and Dee got 55544.
- -> Guy *fell down while running around the south* bend in his last event.
~
~
~
/\ <f.*th>

설명

f로 시작되고 그뒤에 임의의 수의 문자,(.*), th로 끝나는 문자렬이 놓이는 임의의 단어를 탐색한다.

실례 2-12

(Remembered Patterns \ (and \ ).).
% vi textfile (*Before Substitution.*).

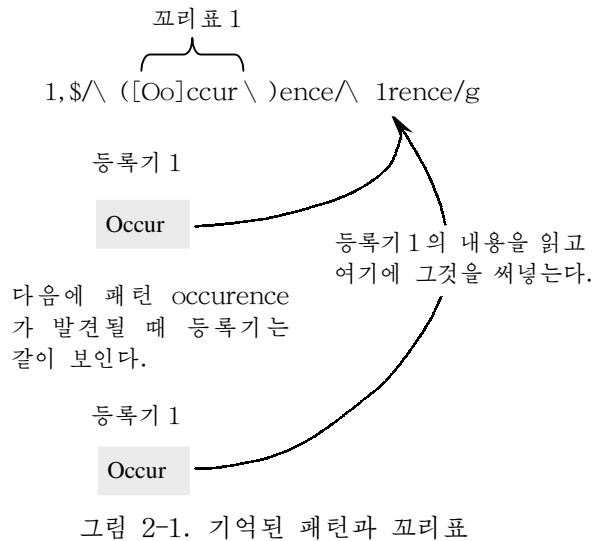
Unusual **occurences** happened at the fair.
Patty won fourth place in the 50 yard dash square and fair.
Occurences like this are rare.
The winning ticket is 55222.
The ticket I got is 54333 and Dee got 55544.
Guy fell down while running around the south bend in his last event.
~
~
~
1. **:l,$s /\ ([Oo]ccur\ )ence /\ lrence /**

% **vi textfile** (After Substitution.).

-->Unusual **occurrences** happened at the fair.
Patty won fourth place in the 50 yard dash square and fair.
--> Occurrences like this are rare.
The winning ticket is 55222.
The ticket I got is 54333 and Dee got 55544.
Guy fell down while running around the south bend in his last event.
~
~

설명

1. 편집기는 전체적인 문자렬 occurence 혹은 Occurrence(주의:단어들의 맞춤법이 틀리게 씌여 진것)를 탐색하여 찾으면 괄호안에 놓인 패턴부분에 꼬리표를 붙인다(즉 occur 혹은 Occur에 꼬리표 붙인다.). 이것은 꼬리표가 붙은 첫 패턴이므로 꼬리표 1이라고 부른다. 패턴은 등록기 1이라고 부르는 기억등록기에 기억된다. 교체가 진행될 때 등록기의 내용은 \ 1로 교체되고 단어의 나머지 rence가 거기에 첨부된다. Occurrence가 occurrence로 되였다(그림 2-1).

그림 2-1. 기억된 패턴과 꼬리표

실례 2-13

% **vi textfile** *(Before Substitution.).*

```
Unusual occurrences happened at the fair.
Patty won fourth place in the 50 yard dash square and fair.
Occurrences like this are rare.
The winning ticket is 55222.
The ticket I got is 54333 and Dee got 55544.
Guy fell down while running around the south bend in his last
event.
~
~
~
```

1. **:s/\ (squrae\ .). and \ (fair\ .)./\ 2 \ 1/**

% **vi textfile** (After Substition.).

```
   Unusual occurrences happened at the fair.
-->Patty won fourth place in the 50 yard dash square and fair.
```

```
 Occurrences like this are rare.
 The winning ticket is 55222.
 The ticket I got is 54333 and Dee got 55544.
Guy fell down while running around the south bend in his last
event.
~
~
~
-------------------------------------------------------------------------------------
```

설명

1. 편집기는 정규식 square와 fair를 탐색하여 square는 1로, fair는 2로 꼬리표를 붙인다. 교체가 진행될 때 등록기 2의 내용은 \ 2로, 등록기 1의 내용은 \ 1로 교체된다(그림 2-2).

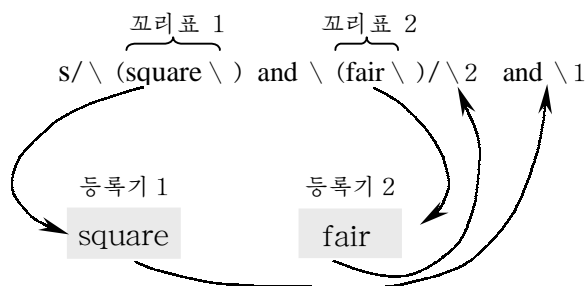



그림 2-2. 여러개의 꼬리표사용

실례 2-14

(Repetition of Patterns (\ {n\ } .).).

```
   % vi textfile
-------------------------------------------------------------------------------------
   Unusual occurrences happened at the fair.
   Patty won fourth place in the 50 yard dash square and fair.
   Occurrences like this are rare.
--> The winning ticket is 55222.
   The ticket I got is 54333 and Dee got 55544.
   Guy fell down while running around the south bend in his last
   event.
   ~
   ~
   ~
1   /5\ {2\ }2\ {3\ }\ . /
-------------------------------------------------------------------------------------
```

설명

1. 2개의 수자 5와 그뒤에 3개의 수자 2 그리고 점을 포함하고 있는 행을 탐색한다.

제 3 장. grep계렬

grep계렬은 지령 grep, egrep, fgrep 등으로 구성된다. grep지령은 전체적으로 파일에서 정규식을 탐색하며 표현을 포함하는 모든 행을 인쇄한다. egrep와 fgrep지령은 grep의 간단한 변종이다. egrep지령은 보다 많은 정규표현메타문자들을 지원하고 있는 확장된 grep이다. fixed grep 또는 fast grep라고 하는 fgrep지령은 모든 기호문자들을 자모처럼 취급한다. 즉 정규표현메타문자들은 특수문자가 아닌데 자기자체를 일치시킨다. 공개쏘프트웨어재단(Free Software Foundation)은 GNU grep라고 하는 grep의 무료판본을 제공하고 있다. 이 grep의 판본은 Linux체계우에서 사용되는 지령이며 Sun의 Solaris조작체계의 /usr/xpq4/bin에서 찾을수 있다. grep의 GNU 판본은 기본정규표현메타문자를 확장하고 POSIX와 호환성을 보장하며 일련의 새로운 지령행추가선택들을 가지고 있다. 또한 완전한 등록부나무구조를 리용하기 위한 rgrep라고 하는 재귀grep를 제공한다.

제 1 절. grep지령

1. grep의 의미

grep라는 이름은 ex편집기에 놓일수 있다. 만일 ex편집기를 사용하여 문자렬을 탐색하려면 ex재촉문에서 다음과 같은 지령을 입력한다.

:/pattern/p

문자렬 pattern을 포함하고 있는 첫 행은 print지령에 의해《p》로 인쇄된다. pattern을 포함하고 있는 모든 행을 인쇄하려면 다음과 같은 지령을 입력할수 있다.

:g/pattern/p

g가 pattern앞에 놓이면 《파일의 모든 행》혹은 《전체적인 교체를 실행한다.》는 것을 의미한다.

탐색패턴을 정규식이라고 하기때문에 pattern을 RE로 교체할수 있으며 지령을 다음과 같이 쓸수 있다.

:g/RE/p

이로부터 grep의 의미와 그 이름의 근원을 알수 있다. 이것은《총체적으로 정규식(RE)을 탐색하며 행을 인쇄(p)한다.》는것을 의미한다. grep를 사용하는데서 좋은 점은 탐색을 실행하기 위한 편집기를 반드시 요구하지 않으며 정규식을 빗선 /안에 놓을 필요가 없다는것이다. 이렇게 하면 ex나 vi를 사용하는것보다 훨씬 더 빠르다.

2. grep는 어떻게 동작하는가

grep지령은 하나 혹은 여러 파일에서 문자들의 패턴을 탐색한다. 패턴이 공백을 포함하면 그것을 인용부호안에 넣어야 한다. 패턴은 인용부호안에 있는 문자렬 즉 한개 단어[1]이며

---

[1]. 단어를 통표라고도 한다.

그뒤에 뒤따르는 모든 다른 단어들은 파일이름으로 취급된다. grep는 그것의 출력을 현시장치에 보내고 입력파일을 변화시키거나 영향을 주지 않는다.

형식

grep word filename filename

실례 3-1

grep Tom /etc/ passwd

설명

grep는 /etc/passwd라고 하는 파일에서 패턴 Tom을 탐색한다. 성공하면 파일로부터 행이 현시장치에 나타난다. 패턴을 찾지 못하면 아무것도 출력하지 않으며 정확한 파일이 아니면 오유가 현시장치에 출력된다. 만일 패턴을 찾으면 grep는 성공을 나타내는 탈퇴상태 령을 되돌려 준다. 패턴을 찾지 못하면 되돌려 지는 탈퇴상태값은 1이며 파일을 찾지 못하면 탈퇴상태값은 2이다.

grep프로그람은 자기의 입력을 파일로부터 받을뿐아니라 표준입력이나 파이프로부터도 받을수 있다. 만일 파일을 지정하지 못하면 grep는 표준입력인 건반으로부터 입력을 받으며 입력이 있을 때까지 대기한다. 지령의 출력이 파이프로부터 들어 오면 grep지령에 대한 입력으로 파이프되며 요구되는 패턴이 일치되면 grep는 현시장치에 출력을 내보낸다.

실례 3-2

% **ps-ef | grep root**

설명

ps지령의 출력(ps-ef는 이 체계상에서 실행하는 모든 프로쎄스를 현시장치에 표시한다.)은 grep에 전송되며 뿌리를 가지고 있는 모든 행들이 인쇄된다.

grep지령은 일련의 정규표현메타문자(표 3-1)를 제공하여 탐색패턴을 더 쉽게 결정할수 있게 한다. 또한 grep지령은 탐색을 실행하거나 행을 표시하는 방법을 변화시키기 위하여 몇개의 추가선택(표 3-2)들을 제공한다. 실례로 추가선택들을 제공하여 행번호표시, 오유만을 표시하는것과 같은 기능들을 해제한다.

표 3-1. grep의 정규표현메타문자

메타문자	기 능	실 례	무엇을 일치시키는가
^	행의 시작문자	'^love'	Love로서 시작하는 모든 행을 일치시킨다.
$	행의 끝문자	'love$ '	Love로서 끝나는 모든 행을 일치시킨다.
.	한개 문자를 일치시킨다.	'l..e'	l을 포함하는 행들과 그뒤에 2개의 문자 그리고 e를 포함하고 있는 모든 행들을 일치시킨다.

(표계속)

메타문자	기 능	실 례	무엇을 일치시키는가
*	령 혹은 그이상의 문자들을 일치시킨다.	‘*love’	령 혹은 그이상의 공백을 가진 행들과 그뒤에 패턴 Love 를 일치시킨다.
[ ]	묶음에서 한개 문자를 일치시킨다.	‘[Ll]ove’	Love 혹은 love를 포함하는 행들을 일치시킨다.
[^]	묶음밖의 한개 문자를 일치시킨다.	‘[^A-K] ove’	A~K사이에 있는 어느 한개 문자를 포함하지 않으면서 그뒤에 ove가 오는 행을 일치시킨다.
\ <	단어시작문자	‘\ <love’	love로 시작하는 한개 단어를 포함하는 행들을 일치시킨다.
\ >	단어끝문자	‘love\ > ’	love로 끝나는 한개 단어를 포함하는 행들을 일치시킨다.
\ (..\ )	꼬리표는 문자들을 일치시킨다.	‘\ (love\ ) ing’	꼬리표는 1로서 문자를 기억하고 있는 등록기의 위치를 표시한다. 문자를 참조하기 위해서는 패턴을 반복하도록 \ 1을 사용한다. 패턴의 제일 왼쪽 부분에서 첫 꼬리표로 시작하여 9개의 꼬리표를 사용할수 있다. 실례로 패턴 ove는 \ 1로서 참조하기 위해 등록기로 패턴 1에 보존한다.
x\ {m\ } x\ {m,\ } x\ {m,n\ }[ㄱ]	문자 x의 반복 즉 m번, 적어도 m번 혹은 m과 n 번사이에서 반복	‘0\ {5\ }’ ‘0\ {5,\ }’ ‘0\ {5,10\ }’	행이 5개의 0을 가지며 적어도 0이 5번 발생하던가 0이 5~10번사이에 발생하면 일치시킨다.

ㄱ. \ {\ }메타문자는 UNIX의 모든 판본들이나 모든 패턴일치편의프로그람에서 지원되지는 않는다. 그것들은 보통 vi와 grep와 함께 리용된다.

표 3-2. grep의 추가선택

추가선택	무엇을 실행하는가
-b	발견되는 블로크번호에 의해 매개 행을 앞에 놓는다. 이것은 때때로 문맥의 디스크블로크번호를 찾을 때 사용한다.
-c	일치되는 행을 표시하는것이 아니라 일치시키는 행을 표시한다.
-h	파일이름을 표시하지 않는다.
-I	비교에서 문자인 경우를 무시한다(즉 대소문자는 구별하지 않는다.).
-n	행바꾸기에 의해 구별되며 일치되는 행을 가진 파일의 이름만을 표시한다.
-s	아무런 결과도 표시하지 않고 실행한다. 즉 오유통보문을 제외하고 아무것도 표시하지 않는다. 이것은 탈퇴상태를 검사하는데 사용된다.

(표계속)

추가선택	무엇을 실행하는가
-v	일치되지 않는 행만을 표시하기 위해 탐색을 반대로 진행한다.
-w	마치도 \ <와 \ >안에 놓인것과 같이 단어로서 표현을 탐색한다. 이것은 grep에서만 사용한다(grep의 모든 판본은 이 특징을 제공하지 않는다. 즉 SCO UNIX에는 이 기능이 없다.).

실례 3-3

% **grep –n '^jack:' /etc/ passwd**

설명

grep는 /etc/passwd파일에서 jack를 탐색한다. 즉 Jack가 행의 시작에 있으면 grep는 jack가 있는 행번호와 행에서 jack의 위치를 인쇄한다.

3. grep와 탈퇴상태

grep지령은 쉘스크립트에서 매우 쓸모 있다. 왜냐하면 그것이 찾으려는 패턴이나 파일을 찾을수 있는가를 나타내는 탈퇴상태를 되돌리기때문이다. 만일 패턴이 발견되면 grep는 탈퇴상태 령을 되돌려 성공을 표시하고 패턴을 찾을수 없으면 그의 탈퇴상태 1을 되돌려 준다. 그리고 파일을 탐색할수 없으면 grep는 탈퇴상태 2를 되돌려 준다(sed와 awk와 같이 패턴을 탐색하는 다른 UNIX편의프로그람들은 패턴을 찾는데서 성공인가 실패인가를 나타내기 위하여 탈퇴상태를 리용하지 않는다. 그것들은 지령에서 문법적오유가 있을 때에만 실패를 알려 준다.).

다음의 실례는 /etc/passwd파일에서 john을 탐색하지 못하는 과정을 보여 준다.

실례 3-4

1. % **grep 'john' /etc/passwd**
2. % **echo $status (csh.).**

   1

   or

   *$ **echo $?** (sh,ksh)*

   1

설명

1. grep는 /etc/passwd파일에서 패턴 john을 탐색하는데 성공하면 0의 상태를 가지고 빠져 나온다. 파일에서 john을 찾지 못하면 grep는 1과 함께 탈퇴한다. 파일을 찾지 못하면 탈퇴상태 2를 되돌린다.
2. C쉘변수 status와 Bourne/Korn쉘변수 ?에는 실행되는 마지막지령의 탈퇴상태가 대입된다.

제 2 절. 정규식을 리용한 grep실례

이 실례에서 사용되는 파일을 datafile이라고 하자.

```
% cat datafile
northwest     NW    Charles Main          3.0    .98    3    34
western       WE    Sharon Gray           5.3    .97    5    23
southwest     SW    Lewis Dalsass         2.7    .8     2    18
southern      SO    Suan Chin             5.1    .95    4    15
southeast     SE    Patricia Hemenway     4.0    .7     4    17
eastern       EA    TB Savage             4.4    .84    5    20
northeast     NE    AM Main Jr            5.1    .94    3    13
north         NO    Margot Weber          4.5    .89    5     9
central       CY    Ann Stephens          5.7    .94    5    13
```

실례 3-5

grep NW datafile

northwest NW Charles Main 3.0 .98 3 34

설명

datafile이라는 파일에서 정규식 NW를 포함하는 모든 행을 인쇄한다.

실례 3-6

grep NW d*

datafile: northwest NW Charles Main 3.0 .98 3 34
db:northwest NW Joel Craig 30 40 5 123

설명

한개 문자 d로 시작하는 모든 파일에서 정규식 NW를 포함하는 모든 행을 인쇄한다. 쉘은 d*를 d로 시작하는 모든 파일로 확장시키는데 이 경우 파일이름은 db와 datafile이다.

실례 3-7

grep '^n' datafile

northwest NW Charles Main 3.0 .98 3 34
northwest NE AM Main Jr. 5.1 .94 3 13
north NO Margot Weber 4.5 .89 5 9

설명

n으로 시작되는 모든 행을 인쇄한다. 탈자기호(^)는 행의 시작문자이다.

실례 3-8

grep '4$' datafile
northwest NW Charles Main 3.0 .98 3 34

설명

4로 끝나는 모든 행을 인쇄한다. 화폐문자($)는 행의 끝문자이다.

실례 3-9

grep TB Savage datafile
grep: Savage: No such file or directory
datafile:eastern EA TB Savage 4.4 .84 5 20

설명

첫 인수가 패턴이고 나머지 모든 인수들은 파일이름이기때문에 grep는 Savage와 datdfile이라는 파일에서 TB를 탐색한다. TB Savage의 탐색은 다음실례를 참고하기 바란다.

실례 3-10

grep 'TB Savage' datafile
eastern EA TB Savage 4.4 .84 5 20

설명

패턴 TB Savage를 포함하고 있는 모든 행을 인쇄한다. 인용부호가 없으면(이 실례에서 단일인용부호 또는 2중인용부호가 있을수 있다.) TB와 Savage사이의 공백은 앞의 실례에서와 같이 grep가 Savage라는 파일과 datafile이라는 파일에서 TB를 탐색하도록 한다.

% cat datafile

Northwest	NW	Charles Main	3.0	.98	3	34
Western	WE	Sharon Gray	5.3	.97	5	23
Southwest	SW	Lewis Dalsass	2.7	.8	2	18
Southern	SO	Suan Chin	5.1	.95	4	15
Southeast	SE	Patricia Hemenway	4.0	.7	4	17
Eastern	EA	TB Savage	4.4	.84	5	20
Northeast	NE	AM Main Jr	5.1	.94	3	13

North	NO	Margot Weber	4.5	.89	5	9
Central	CY	Ann Stephens	5.7	.94	5	13

실례 3-11

grep ' 5 \ . . ' datafile

western	*WE*	*Sharon Gray*	*5.3*	*.97*	*5*	*23*
southern	*SO*	*Suan Chin*	*5.1*	*.95*	*4*	*15*
northeast	*NE*	*AM Main Jr*	*5.1*	*.94*	*3*	*13*
central	*CT*	*Ann Stephens*	*5.7*	*.94*	*5*	*13*

설명

수자 5와 그뒤에 점 그리고 임의의 한개 문자를 포함하는 행을 출력한다. 메타문자"."은 거꿀빗선(\ )에 의해 탈퇴되지 않으면 한개 문자를 표시한다. 탈퇴될 때 그 문자는 특수한 메타문자로 되지 않으며 문자 그자체 즉 점을 나타낸다.

실례 3-12

grep ' \ .5 ' datafile

north	NO	Margot Weber	4.5	.89	5	9

설명

표현 5를 가지는 임의의 행을 표시한다.

실례 3-13

grep ' ^[we] ' datafile

western	*WE*	*Sharon Gray*	*5.3*	*.97*	*5*	*23*
eastern	*EA*	*TB Savage*	*4.4*	*.84*	*5*	*20*

설명

w나 e로 시작하는 행을 인쇄한다. 탈자기호(^)는 행시작문자이며 중괄호안에 있는 어느 한개 문자가 일치된다.

실례 3-14

grep ' [^0-9] ' datafile

northwest	*NW*	*Charles Main*	*3.0*	*98*	*3*	*34*
western	*WE*	*Sharon Gray*	*5.3*	*.97*	*5*	*23*
southwest	*SW*	*Lewis Dalsass*	*2.7*	*.8*	*2*	*18*
southern	*SO*	*Suan Chin*	*5.1*	*.95*	*4*	*15*
southeast	*SE*	*Patricia Hemenway*	*4.0*	*.7*	*4*	*17*

```
eastern      EA    TB Savage        4.4   .84   5   20
northeast    NE    AM Main Jr       5.1   .94   3   13
north        NO    Margot Weber     4.5   .89   5   9
central      CT    Ann Stephens     5.7   .94   5   13
```

설명

한개 수자를 포함하지 않는 모든 행을 인쇄한다. 모든 행에 한개의 수자도 없기때문에 모두 인쇄된다(-v추가선택을 참고).

실례 3-15

```
grep '[A-Z][A-Z] [A-Z]' datafile
eastern      EA    TB Savage        4.4   .84   5   20
northeast    NE    AM Main Jr       5.1   .94   3   13
```

설명

2개의 대문자뒤에 한개의 공백과 대문자가 놓이는 모든 행을 인쇄한다. 즉 TB Savage와 AM Main을 인쇄한다.

실례 3-16

```
grep   'ss* ' datafile
northwest    NW    Charles Main     3.0   98    3   34
southwest    SW    Lewis Dalsass    2.7   .8    2   18
```

설명

문자 s와 그뒤에 임의의 수로 련속적으로 놓이는 문자 s와 한개의 공백이 놓이는 모든 행을 인쇄한다. Charles와 Dalsass를 탐색한다.

```
% cat datafile
northwest    NW    Charles Main         3.0   .98   3   34
western      WE    Sharon Gray          5.3   .97   5   23
southwest    SW    Lewis Dalsass        2.7   .8    2   18
southern     SO    Suan Chin            5.1   .95   4   15
southeast    SE    Patricia Hemenway    4.0   .7    4   17
eastern      EA    TB Savage            4.4   .84   5   20
northeast    NE    AM Main Jr           5.1   .94   3   13
north        NO    Margot Weber         4.5   .89   5    9
central      CY    Ann Stephens         5.7   .94   5   13
```

실례 3-17

```
grep '[a-z]\ {9\ }' datafile
northwest   NW   Charles Main         3.0   98    3   34
southwest   SW   Lewis Dalsass        2.7   .8    2   18
southeast   SE   Patricia Hemenway    4.0   .7    4   17
northeast   NE   AM Main Jr           5.1   .94   3   13
```

설명

적어도 9개의 소문자가 련속적으로 놓이는 모든 행을 인쇄한다. 실례로 northwest, southwest, southeast, northeast를 인쇄한다.

실례 3-18

```
grep '\ (3\ .).\ .[0-9].*\ 1        *\ 1' datafile
northwest   NW   Charles Main         3.0   98    3   34
```

설명

수자 3의 뒤에 점과 다른 수자, 임의의 수의 문자들(.*), 다른 수자 3(초기에 꼬리표가 붙은), 임의의 수의 타브들 그리고 또 다른 수자 3이 놓이는 행을 인쇄한다. 3이 괄호\ (3\ )안에 있기때문에 후에 \ 1로서 참조될수 있다. \ 1은 이것이 \ (\ )쌍으로 꼬리표가 붙은 첫번째 표현이였다는것을 의미한다.

실례 3-19

```
grep ' \ < north '  datafile
northwest   NW   Charles Main         3.0   98    3   34
northeast   NE   AM Main Jr           5.1   .94   3   13
north       NO   Margot Weber         4.5   .89   5   9
```

설명

north로 시작되는 단어를 가진 모든 행을 인쇄한다. \ <는 단어시작문자이다.

실례 3-20

```
grep ' \ <north '  datafile
north       NO   Margot Weber         4.5   .89   5   9
```

설명

단어 north를 포함하면 행을 인쇄한다. \ <는 단어의 시작문자이며 \ > 는 단어의 끌문자이다.

실례 3-21

```
grep   '\ <[a-z].*n\ >'   datafile
northwest   NW   Charles Main    3.0   98   3   34
western     WE   Sharon Gray     5.3   .97  5   23
southern    SO   Suan Chin       5.1   .95  4   15
eastern     EA   TB Savage       4.4   .84  5   20
northeast   NE   AM Main Jr      5.1   .94  3   13
central     CT   Ann Stephens    5.7   .94  5   13
```

설명

한개 소문자로 시작되고 그뒤에 임의의 개수문자가 놓이는 한개 단어와 n으로 끝나는 한개 단어를 가지는 모든 행을 인쇄한다. .*기호를 주의하여야 한다. 이것은 공백을 비롯한 임의의 문자를 의미한다.

제 3 절. 파이프를 리용한 grep

흔히 grep는 파일로부터 입력을 받지 않고 파이프로부터 입력을 받는다.

실례 3-22

```
% ls –1
drwxrwxrwx   2   ellie   2441   Jan   6   12:34   dirl
-rw-r--r--   1   ellie   1538   Jan   2   15:50   filel
-rw-r--r--   1   ellie   1539   Jan   3   13:36   file2
drwxrwxrwx   2   ellie   2341   Jan   6   12:34   grades

% ls –1 | grep '^d'
drwxrwxrwx   2   ellie   2441   Jan   6   12:34   dir1
drwxrwxrwx   2   ellie   3341   Jan   6   12:34   grades
```

설명

ls지령의 출력은 grep에 파이프된다. d로 시작하는 모든 출력행들이 인쇄된다. 즉 모든 등록부가 인쇄된다.

제 4 절. 추가선택을 리용한 grep

grep지령은 동작을 조종하는 추가선택들을 가지고 있다. UNIX의 모든 판본들이 꼭같은 추가선택들을 지원하지는 않기때문에 전체 목록에 대한 안내페지를 정확히 검사하여야 한다.

```
% cat datafile
northwest   NW   Charles Main   3.0   .98   3   34
```

western	WE	Sharon Gray	5.3	.97	5	23
southwest	SW	Lewis Dalsass	2.7	.8	2	18
southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Patricia Hemenway	4.0	.7	4	17
eastern	EA	TB Savage	4.4	.84	5	20
northeast	NE	AM Main Jr	5.1	.94	3	13
north	NO	Margot Weber	4.5	.89	5	9
central	CY	Ann Stephens	5.7	.94	5	13

실례 3-23

grep –n '^south' datafile

```
3:southwest    SW   Lewis Dalsass       2.7   .8    2    18
4:southern     SO   Susan Chin          5.1   .95   4    15
5:southeast    SE   Patricia Hemenway   4.0   .7    4    17
```

설명

-n추가선택은 매 행의 앞에 패턴이 발견된 행의 번호를 붙이고 그다음 행을 놓는다.

실례 3-24

grep –i 'pat' datafile

```
southeast    SE   Patricia Hemenway   4.0   .7   4    17
```

설명

-i추가선택은 대소문자를 구별하지 않게 한다. 표현 pat는 소문자와 대문자가 임의로 조합될수 있다.

실례 3-25

grep -v 'Suan Chin' datafile

```
northwest   NW   Charles Main        3.0   98    3    34
western     WE   Sharon Gray         5.3   .97   5    23
southwest   SW   Lewis Dalsass       2.7   .8    2    18
southeast   SE   Patricia Hemenway   4.0   .7    4    17
eastern     EA   TB Savage           4.4   .84   5    20
northeast   NE   AM Main Jr          5.1   .94   3    13
north       NO   Margot Weber        4.5   .89   5    9
central     CT   Ann Stephens        5.7   .94   5    13
```

설명

여기서 -v추가선택은 패턴 Suan Chin을 포함하지 않는 모든 행을 인쇄한다. 이 추가선택은 입력파일로부터 한개의 지정된 입구점을 지우기할 때 사용된다. 입구점을 실지로 지우기하기 위해서는 grep의 출력을 림시파일에 방향바꾸기한 다음에 아래에 보여 준바와 같이 림시파일의 이름을 초기파일이름으로 다시 변경시킨다.

```
grep -v 'Suan Chin'datafile>temp
mv temp datafile
```

datafile로부터 출력을 방향바꾸기할 때 림시파일을 사용한다는것을 명심해야 한다. dataile로부터 datafile로 방향바꾸기하면 쉘은 datafile을 《지우기》한다(22페지의 《방향바꾸기》를 보기 바란다.).

% cat datafile

northwest	NW	Charles Main	3.0	.98	3	34
western	WE	Sharon Gray	5.3	.97	5	23
southwest	SW	Lewis Dalsass	2.7	.8	2	18
southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Patricia Hemenway	4.0	.7	4	17
eastern	EA	TB Savage	4.4	.84	5	20
northeast	NE	AM Main Jr	5.1	.94	3	13
north	NO	Margot Weber	4.5	.89	5	9
central	CY	Ann Stephens	5.7	.94	5	13

실례 3-26

grep –1 ' SE ' *
datafile
datebook

설명

-l추가선택은 grep가 패턴이 본문의 행대신에 놓이는 파일이름만을 인쇄하게 한다.

실례 3-27

grep –c 'west' datafile
3

설명

-c추가선택은 패턴이 발견된 행의 개수를 출력하기 위하여 grep를 기동시킨다. 이것은 패턴의 발생수를 의미하지 않는다. 실례로 west가 행에서 3번 탐색되면 한번만 행을 계수한다.

실례 3-28

grep –w 'north' datafile

north NO Margot Weber 4.5 .89 5 9

설명

-w추가선택은 단어의 부분이 아니라 단어[2]일 때만 grep가 패턴을 탐색하게 한다. 단어 north를 포함하는 행만을 출력한다. 즉 northwest는 인쇄되지 않는다.

실례 3-29

echo $LOGNAME

lewis

grep –i "$LOGNAME" datafile

southwest SW Lewis Dalsass 2.7 .8 2 18

설명

쉘의 ENV변수의 값인 LOGNAME이 인쇄된다. 이것은 사용자의 등록가입이름을 포함한다. 만일 변수가 2중인용부호안에 있으면 쉘에 의해 확장되며 변수에 한개이상의 단어가 대입되는 경우 공백은 쉘해석으로부터 보호된다. 만일 단일인용부호가 사용되면 변수교체는 일어 나지 않는다. 즉 $LOGNAME이 인쇄된다.

1. grep 개괄

표 3-3은 grep지령의 실례들과 그것들이 무엇을 수행하는가를 보여 주고 있다.

표 3-3. grep의 개괄

grep지령	무엇을 수행하는가
grep '\ <Tom \ >'file	단어 Tom을 포함하고 있는 행을 인쇄한다.
grep 'Tom Savage'file	Tom Savage를 포함하는 행을 인쇄한다.
grep '^Tommy'file	Tommy가 행의 시작에 놓이면 행을 인쇄한다.
grep '\ .bak$'file	.bak.으로 끝나는 행을 인쇄한다. 한개 인용부호는 해석으로부터 화페문자($)를 보호한다.
grep '[Pplyramid'*	현재작업등록부의 모든 파일에서 Pyramid를 포함하고 있는 행들을 인쇄한다.
grep '[A-Z]'file	적어도 한개 대문자를 포함하고 있는 행들을 인쇄한다.

2. 단어는 행의 머리부에서 시작되거나 공백이 앞에 놓이고 공백, 점 혹은 행바꾸기로 끝나는 자모와 수자로 된 문자들의 렬이다.

(표계속)

grep지령	무엇을 수행하는가
grep '[0-9]'file	적어도 한개 수자를 포함하고 있는 행들을 인쇄한다.
grep '[A-Z]...[0-9]'file	대문자로 시작하고 수자로 끝나는 5개의 문자패턴을 포함하고 있는 행들을 인쇄한다.
grep- w'[tT]est'file	단어 Test와 test를 가진 행들을 인쇄한다.
grep-s "Mark Todd"file	Mark Todd를 포함하고 있는 행들을 탐색하지만 인쇄하지는 않는다. grep의 탈퇴상태를 검사할 때 사용할수 있다.
grep-v 'Mary'file	Mary를 포함하지 않는 모든 행들을 인쇄한다.
grep-I'sam'file	여러가지 조합의 sam 즉 SAM, sam, Sam, sAm 등을 포함하고 있는 모든 파일이름들을 인쇄한다.
grep-l'Dear Boss'*	Dear Boss를 포함하고 있는 모든 파일이름들을 목록에 기입한다.
grep-nTom'file	행번호를 가진 행들을 일치시키는것을 선행시킨다.
grep "$name"file	변수 name의 값을 전개하고 그 값을 포함하고 있는 모든 행들을 인쇄한다. 2중인용부호를 사용하여야 한다.
grep '$5'file	문자 $5를 포함하고 있는 행들을 인쇄한다. 한개 인용부호를 사용하여야 한다.
ps-ef|grep "^*user1"	행의 시작에 공백이 없거나 있을수도 있는 user1을 탐색하면서 grep에 ps-ef의 출력을 련결한다.

제 5 절. egrep(확장된 grep)

egrep를 사용하는데서 기본우점은 추가적인 정규표현메타문자(표 3-4)들이 grep에서 제공되는 모임에 추가된것이다. 그러나 \ (\ )와 \ {\ }는 허용되지 않는다(Linux를 사용하려면 GNU grep-E를 보면 된다.).

표 3-4. egrep의 정규표현메타문자

메타문자	기 능	실 례	무엇을 일치시키는가
^	행의 시작문자	'^love'	love로서 시작하는 모든 행을 일치시킨다.
$	행의 끝문자	'love$ '	love로서 끝나는 모든 행을 일치시킨다.
.	한개 문자를 일치시킨다.	'l..e'	1과 그뒤로 2개의 문자, 한개 e가 놓이는 단어를 포함하고 있는 행들을 일치시킨다.
*	령 혹은 그이상의 문자들을 일치시킨다.	'*love'	령 혹은 그이상의 공백뒤에 패턴 love가 놓이는 행들을 일치시킨다.

(표계속)

메타문자	기 능	실 례	무엇을 일치시키는가
[ ]	묶음에서 한개 문자를 일치시킨다.	'[Ll]ove'	Love 혹은 love를 포함하는 행들을 일치시킨다.
[^]	묶음밖의 한개 문자를 일치시킨다.	'[^A-KM-Z]ove'	A에서 K까지와 M에서 Z 그리고 뒤에 행들을 ove가 놓이는 단어를 포함하지 않는 행들을 일치시킨다.
egrep의 새로운 메타문자 :			
+	하나 혹은 그이상의 선행한 문자들을일치시킨다.	'[a-z]+ove'	하나 혹은 그이상의 소문자와 그뒤에 ove가 놓이는 단어들을 일치시킨다. Move, approve, love, behoove 등이 탐색된다.
?	령 혹은 한개 선행한 문자들을 일치시킨다.	'lo?ve'	L과 그뒤에 한개 소문자 o를 일치시킨다.
a|b	a 혹은 b를 일치시킨다.	'love|hate'	표현 love 또는 hate를 일치시킨다.
()	그룹문자	'love(ablelly.).	lovable 혹은 lovely를 일치시킨다. 하나 혹은 그이상의 ove의 발생을 일치시킨다.

1. egrep의 실례

다음의 실례는 정규표현메타문자의 새로 확장된 지령묶음을 egrep에서 사용하는 방법을 설명하였다. 이미전에 서술된 grep실례들은 표준메타문자들의 사용에 대하여 설명하였는데 egrep와 같은 방법으로 동작한다. Egrep 역시 grep처럼 지령행에서 동일한 추가선택을 사용한다.

실례 3-30

egrep 'NW|EA' datafile

```
northwest    NW    Charles Main    3.0    .99    3    34
eastern      EA    TB Savage       4.4    .84    5    20
```

설명

표현 NW나 EA를 포함하면 행을 출력한다.

```
% cat datafile
northwest    NW    Charles Main    3.0    .98    3    34
western      WE    Sharon Gray     5.3    .97    5    23
```

southwest	SW	Lewis Dalsass	2.7	.8	2	18
southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Patricia Hemenway	4.0	.7	4	17
eastern	EA	TB Savage	4.4	.84	5	20
northeast	NE	AM Main Jr	5.1	.94	3	13
north	NO	Margot Weber	4.5	.89	5	9
central	CY	Ann Stephens	5.7	.94	5	13

실례 3-31

egrep ‘ 3+ ’ datafile

northwest	*NW*	*Charles Main*	*3.0*	*.98*	*3*	*34*
western	*WE*	*Sharon Gray*	*5.3*	*.97*	*5*	*23*
northeast	*NE*	*AM Main*	*5.1*	*.94*	*3*	*13*
cntral	*CT*	*Ann Stephens*	*5.7*	*.94*	*5*	*13*

설명

하나 혹은 그이상의 수자 3을 가진 모든 행을 출력한다.

실례 3-32

egrep ‘2\ >? ’ [0-9]’ datafile

western	*WE*	*Sharon Gray*	*5.3*	*.97*	*5*	*23*
southwest	*SW*	*Lewis Dalsass*	*2.7*	*.8*	*2*	*18*
eastern	*EA*	*TB Savage*	*4.4*	*.84*	*5*	*20*

설명

한개 수자 2와 그뒤에 령 혹은 점 그리고 한개 수자가 놓이는 모든 행을 인쇄한다.

실례 3-33

egrep ‘(no)+’ datafile

northwest	*NW*	*Charles Main*	*3.0*	*.98*	*3*	*34*
northeast	*NE*	*AM Main*	*5.1*	*.94*	*3*	*13*
north	*NO*	*Margot Weber*	*4.5*	*.89*	*5*	*9*

설명

패턴그룹 no가 하나이상 련속적으로 발생하는 행들을 인쇄한다.

실례 3-34

```
egrep 'S(h|u)'  datafile
Western     WE     Sharon  Gray     5.3   .97   5   23
Southern    SO     Susan Chin       5.1   .95   4   15
```

설명

s와 그뒤에 h나 u가 놓이는 모든 행을 인쇄한다.

실례 3-35

```
egrep 'Sh|u'  datafile
western     WE     Sharon  Gray          5.3   .97   5   23
southern    SO     Suan   Chin           5.1   .95   4   15
southwest   SW     Lewis Dalsass         2.7   .8    2   18
southeast   SE     Patricia Hemenway     4.0   .7    4   17
```

설명

표현 Sh 혹은 u를 포함하는 모든 행을 인쇄한다.

2. egrep개괄

표 3-5는 egrep지령들과 그것들이 어떤 기능을 수행하는가를 보여 주었다.

표 3-5. egrep의 개괄

egrep지령	무엇을 수행하는가
egrep '^+'file	하나이상의 공백을 가지고 시작되는 행들을 인쇄한다.
egrep '^*'file	령 혹은 그이상의 공백을 가지고 시작하는 행들을 인쇄한다.ㄱ
egrep '(Tom|Dan.).Savage'file	Tom Savage 혹은 Dan Savage를 포함하고 있는 행들을 인쇄한다.
egrep '(ab)+file	ab가 하나이상 존재하는 행들을 인쇄한다.
egrep '*X[0-9]?'file	X와 령 혹은 한개 단일한 수자로 시작되는 행들을 인쇄한다.
egrep 'fun\ .$'*	모든 파일에서 fun.으로 끝나는 행들을 인쇄한다.ㄱ
egrep 'A-Z]+'file	하나이상의 대문자를 포함하는 행들을 인쇄한다.ㄱ
egrep '[0-9]'file	한개 수자를 포함하고 있는 행들을 인쇄한다.ㄱ
egrep '[A-Z]...[0-9]'file	한개 대문자와 그뒤로 3개의 임의의 문자로 시작되고 한개 수자로 끝나는 5개의 문자패턴을 포함하고 있는 행들을 인쇄한다.ㄱ

(표계속)

egrep지령	무엇을 수행하는가
egrep '[tT]est'files	Test 혹은 test를 가진 행들을 인쇄한다.
egrep "Susan Jean"file	Susan Jean을 포함하고 있는 행들을 인쇄한다.ㄱ
egrep –v 'Mary'file	Mary를 포함하지 않는 모든 행들을 인쇄한다.ㄱ
egrep =I 'sam'file	sam의 다양한 형태 즉 SAM, sam, SaM, sAm 등을 포함하고 있는 모든 행들을 인쇄한다.ㄱ
egrep –l 'Dear Boss'*	Dear Boss를 포함하고 있는 모든 파일이름들을 목록에 기입한다.ㄱ
egrep –n 'Tom'file	행번호로 일치되는 행들을 앞에 놓는다.ㄱ
egrep –s "$name"file	변수 name을 확장하고 그것을 탐색하지만 아무것도 인쇄하지는 않는다. egrep의 탈퇴상태를 검사하기 위해 리용할수 있다.

ㄱ·egrep와 grep는 같은 방법으로 이 패턴을 처리한다.

제 6 절. 고정grep 혹은 고속grep

fgrep지령은 grep처럼 동작하지만 임의의 정규표현메타문자들이 특수문자로 될 때 그것을 인식하지 않는다.

모든 문자들은 그자체만을 표현한다. 탈자문자는 단순히 그자체이며 화페기호도 그자체이다(Linux를 사용하려면 GNU grep-F를 보면 된다.).

실례 3-36

```
% fgrep ' [A-Z]****[0-9]..$5.00 '  file
```

설명

문자렬 [A-Z]****[0-9]..$5.00을 포함하고 있는 파일에서 모든 행을 탐색한다. 모든 문자들은 그자체로서 취급된다. 특수한 문자들이 없다.

UNIX도구들에 대한 련습

련습문제 1: grep련습

Steve Blenheim 238-923-7366 95 Latham Lane, Easton, PA 83755 11/12/56:20300
Betty Boop 245-836-8357 635 Cutesy Lane, Hollywood, CA 91464 6/23/23:14500
Igor Chevsky 385-375-8395 3567 Populus Place, Caldwell, NJ 23875 6/18/68 :23400

Norma Corder 397-857-2735 74 Pine Street, Dearborn, MI 23874 3/28/45:245700
Jennifer Cowan 548-834-2348 583 Laurel Ave, Kmgsville, TX 83745 10/1/35 :58900
Jon DeLoach 408-253-3122 123 Park St, San Jose, CA 04086 7/25/53:85100
Karen Evich 284-758-2857 23 Edgechff Place, Lincoln, NB 92743 7/25/53:85100
KarenEvich 284-758-2867 23 Edgechff Place, Lincoln, NB 92743 11/3/35:58200
Karen Evich 284-758-2867 23 Edgechff Place, Lincoln, NB 92743 11/3/35:58200
Fred Fardbarkle 674-843-1385 20 Parak Lane, DeLuth, MN 23850 4/12/23:780900
Fred Fardbarkle 674-843-1385 20 Parak Lane, DeLuth, MN 23850 4/12/23:780900
Lori Gortz 327-832-5728 3465 Mirlo Street, Peabody, MA 34756 10/2/65:35200
Paco Gutierrez 835-365-1284 454 Easy Street, Decatur, IL 75732 2/28/53:123500
Ephram Hardy 293-259-5395 235 CarltonLaneJohet, IL 73858 8/12/20:56700
James Ikeda 834-938-8376 23445 Aster Ave , Allentown, NJ 83745 12/1/38:45000
Barbara Kertz 385-573-8326 832 Ponce Drive, Gary, IN 83756 12/1/46:268500
Lesley Kirstm 408-456-1234 4 Harvard Square, Boston, MA 02133 4/22/62:52600
William Kopf 846-836-2837 6937 Ware Road, Milton, PA 93756 9/21/46:43500
Sir Lancelot 837-835-8257 474 Camelot Boulevard, Bath, WY 28356 5/13/69:24500
Jesse Neal 408-233-8971 45 Rose Terrace, San Francisco, CA 92303 2/3/36:25000
Zippy Pmhead 834-823-8319 2356 Bizarro Ave , Farmount, IL 84357 1/1/67:89500
Arthur Putie 923-835-8745 23 Wimp Lane, Kensington, DL 38758 8/31/69:126000
Popeye Sailor 156-454-3322 945 Bluto Street,Anywhere, USA 29358 3/19/35:22350
Jose Santiago 385-898-8357 38 Fife Way, Abilene, TX 39673 1/5/58:95600
Tommy Savage 408-724-0140 1222 OxbowCourt,Sunnyvale,CA 94087 5/19/66 :34200
Yukio Takeshida 387-827-1095 13 Uno Lane, Ashville, NC 23556 7/1/29:57000
Vinh Tranh 438-910-7449 8235 Maple Street, Wilmmgton, VM 29085 9/23/63:68900

(CD의 databook라는 자료기지를 참고)

1. 문자렬 San을 포함하고 있는 모든 행을 인쇄하시오.
2. 사람이름의 첫 문자가 J인 모든 행을 인쇄하시오.
3. 700에서 끝나는 모든 행을 인쇄하시오.
4. 834를 포함하지 않는 모든 행을 인쇄하시오.
5. 생일이 December인 모든 행을 인쇄하시오.
6. 전화번호가 408구역코드인 모든 행을 인쇄하시오.
7. 대문자를 포함하여 그뒤에 4개의 소문자, 한개 반점, 한개 공백, 한개 대문자를 포함하고 있는 모든 행을 인쇄하시오.
8. 이름의 마지막문자가 K 혹은 k로 시작하는 행을 인쇄하시오.
9. 로임이 여섯자리인 행번호앞에 있는 행을 인쇄하시오.
10. Lincoln이나 lincoln을 포함하고 있는 행을 인쇄하시오(grep가 경우에 따라 인식할수 없다는것을 주의하시오.).

제 4 장. 흐름선편집기 sed

제 1 절. sed란 무엇인가

sed는 흐름선의 비대화형편집기이다. 이 편집기로는 vi와 ex편집기에서 리용되는 같은 종류의 편집과제를 수행할수 있다. sed프로그람은 편집기와 대화형으로 작업하지 않고 지령행에 편집지령을 입력하고 파일들을 이름 지으며 현시장치에서 편집지령의 출력을 볼수 있게 한다. sed편집기는 비파괴적인 편집기이다. 이 편집기는 쉘방향바꾸기로 출력을 기억하지 않는한 사용자파일을 변경시키지 못한다. 모든 행은 기정값에 의하여 현시장치에 인쇄된다.

sed편집기는 쉘스크립트들에서 쓸모가 있는데 만일 스크립트에서 vi나 ex와 같은 대화형편집기를 사용하면 스크립트의 사용자는 편집기에 정통해야 하며 열린 파일에 대한 수정은 사용자의 의도와는 다르게 진행될수 있다. 다중편집이 필요하거나 쉘지령행[1]에서 sed지령을 인용부호에 넣는것이 시끄러우면 sed스크립트라는 파일에 sed지령을 써넣을수 있다.

제 2 절. sed는 어떻게 동작하는가

sed편집기는 한번에 한개 파일(혹은 입력)의 한 행을 처리하며 현시장치에 그것을 보낸다. 이 편집기지령들은 vi와 ed/ex편집기로 부터 인식할수 있는것들이다. sed편집기는 패턴공간이라고 하는 림시완충기에 현재 처리하는 행을 기억한다. 일단 sed가 패턴공간에서 행의 프로쎄스를 끝내면(즉 그 행에서 sed지령의 실행이 끝나면) 패턴공간의 행은 현시장치에 보내진다(지령이 행을 지우거나 그의 인쇄를 정지시키지 않으면). 행이 처리된 다음 패턴공간으로부터 행은 지워 지며 다음행이 패턴공간으로 읽어 지고 처리되며 표시된다. sed는 입력파일의 마지막행이 처리되면 완료된다. 림시완충기에 매 행이 기억되고 그 행에서 편집이 실행되기때문에 초기파일은 결코 변화되거나 파괴되지 않는다.

제 3 절. 주소화

편집하려고 하는 행을 결정하기 위해 주소화를 사용할수 있다. 주소는 수자형태, 정규식 혹은 이 2개의 조합이 될수 있다. sed는 주소를 규정하지 않고도 입력파일의 모든 행을 처리한다.

주소가 한개 수자로 구성될 때 수자는 행번호를 표시한다. 화폐문자는 입력파일의 마지막행을 표시하기 위해 사용될수 있다. 만일 한개 반점이 두 행번호를 분리하면 처리될수 있는 주소는 첫번째와 마지막행을 포함하여 행들의 범위내에 있다. 범위는 수자, 정규식 혹은 수자와 정규식의 조합이 될수 있다. sed지령들은 sed에게 행에서 무엇을 해야 하는가를 알려 준다. 즉 sed는 행을 인쇄, 지우기, 변경 등을 한다.

---

1. 쉘은 지령이 지령행에 입력될 때 임의의 메타문자나 공백을 해석한다는것을 명심해야 한다. 쉘에 의해서 해석되는 sed 지령의 임의의 문자들은 인용부호안에 넣어야 한다.

형식

Sed '지령' 파일이름(들)

실례 4-1

1. % **sed '1,3p' myfile**
2. % **sed –n '/[Jj]ohn/p'   datafile**

설명

1. myfile의 1행부터 3행까지를 인쇄한다.
2. myfile에서 패턴 John이나 john과 일치되는 행만을 인쇄한다.

제 4 절. 지령과 추가선택

sed지령은 주소에 의해 지정되는 매 행을 어떻게 처리하는가를 sed에 알려 준다. 만일 주소가 없으면 sed는 매 입력행을 처리한다(%는 csh재촉문이다.). sed지령의 형태와 그 기능을 표 4-1에 보여 주었다. 추가선택목록과 매 추가선택이 sed의 동작을 어떻게 조종하는가에 대해서는 표 4-2에 보여 주었다.

실례 4-2

% sed '1, 3d' file

설명

sed는 1행부터 3행까지 지운다.

표 4-1. sed지령

지 령	기 능
d\	하나이상의 본문행을 현재행에 추가한다.
c\	새 본문으로 현재행의 본문을 변화(교체)시킨다.
d	행을 지운다.
i\	현재행에 본문을 삽입한다.
h	보존완충기에 패턴공간의 내용을 복사한다.
H	보존완충기에 패턴공간의 내용을 추가한다.
g	보존완충기에 있는 내용을 읽고 거기에 있는 내용들을 추가하면서 패턴완충기에 그것을 복사한다.
G	보존완충기에 있는 내용을 읽고 거기에 있는 내용들을 추가하면서 패턴완충기에 그것을 복사한다.
l	인쇄되지 않는 문자들을 목록에 삽입한다.
p	행들을 인쇄한다.

(표계속)

지 령	기 능
n	다음입구점을 읽고 첫 지령보다 오히려 더 좋은 다음지령을 가진 행바꾸기를 처리하기 시작한다.
q	sed에서 탈퇴한다.
r	파일에서 행들을 읽는다.
!	선택된 한개 행을 제외하고 모든 행의 지령을 실행한다.
S	한개 문자렬을 교환한다.
교환기발:	
g	행에서 총체적으로 교환한다.
p	행을 인쇄한다.
w	파일에 행들을 쓰기한다.
x	패턴공간을 가진 보존공간의 내용을 교환한다.
Y	한개 문자를 다른것으로 변형시킨다(y와 함께 정규식문자들을 사용할수 있다.)

표 4-2. sed추가선택

추가선택	기 능
-e	다중편집을 허용한다.
-f	sed스크립트파일이름을 앞부분에 놓는다.
-n	기정값출력을 중지한다.

다중지령을 사용하거나 주소들을 그 주소의 범위내에서 중복시킬 필요가 있을 때 지령들은 대괄호안에 놓이게 되며 매 지령은 분리된 행에 놓이거나 또는 반두점으로 끝난다. 감탄부호(!)는 지령행을 무효로 선언하기 위해 사용된다. 실례로

% sed '/Tom/d'file

은 Tom을 포함하는 모든 행을 지우도록 sed에 알려 주며 반대로

% sed '/Tom/!d'file

은 Tom을 포함하지 않는 행을 지우도록 sed에 알려 준다.

sed의 추가선택은 -e, -f, -n이다. -e는 지령행에서 다중편집을 위해 사용되며 -f는 sed스크립트파일이름앞에 놓이며 –n은 출력인쇄를 금지시킨다.

제 5 절. 오유통보문과 탈퇴상태

sed는 문법오유가 있으면 간단한 오유통보문을 표준오유로 내보낸다. 그러나 sed가 오유를 찾을수 없으면 그것은 자기 기능을 수행할수 없다. sed가 쉘에 되돌려 주는 탈퇴상태는 문법에 오유가 없을 때 성공에 대해서는 령으로 되며 실패에 대해서는 령이 아닌 옹근수로 된다.[2]

실례 4-3

1. % **sed '1,3v ' file**
   sed: Unrecognized command: 1,3v
   % **echo $status (echo $?** *if using Korn or Bourne shell)*
   2
2. % **sed '/^>Tohn' tile**
   sed: Illegal or missing delimiter: /^John
3. % **sed 's /134345/g' file**
   sed: Ending delimiter missing on substitution: s/134345/g

설명

1. v지령은 sed에 의해서는 인식되지 않는다. 탈퇴상태가 2로서 sed가 문법적 오유때문에 탈퇴했다는것을 반영한다.
2. 패턴 /^John은 닫는 빗선기호 /를 요구한다.
3. 교환지령 s는 탐색문자렬을 포함하고 있으나 교체문자렬을 포함하지는 않는다.

1. 메타문자

sed도 grep와 같이 패턴탐색을 조종하기 위한 많은 특정의 메타문자들을 지원하고 있다. 이것을 표 4-3에 보여 주었다.

표 4-3. sed의 정규표현메타문자

메타문자	기 능	실 례	무엇을 일치시키는가
^	행의 시작문자	0/^love/	love로서 시작하는 모든 행을 일치시킨다.
$	행의 끝문자	/love$/	love로서 끝나는 모든 행을 일치시킨다.
.	한개 문자를 일치시키거나 행바꾸기는 없다.	/l..e/	l을 포함하는 행들과 그뒤에 2개의 문자 그리고 e를 포함하고 있는 행들을 일치시킨다.

2. 완전한 진단목록에 대해서는 sed 에 대한 UNIX 안내페지를 보면 된다.

(표계속)

메타문자	기 능	실 례	무엇을 일치시키는가
*	령 혹은 그이상의 문자들을 일치시킨다.	/*love/	령 혹은 그이상의 공백과 그뒤에 패턴 love 를 가진 행들을 일치시킨다.
[ ]	묶음에서 한개 문자를 일치시킨다.	/[Ll]ove/	Love 혹은 love를 포함하는 행들을 일치시킨다.
[^]	묶음에서 한개 문자를 포함하지 않는것을 일치시킨다.	/[^A-KM-Z]ove/	love 혹은 Love를 포함하는 행들을 범위 A~K 혹은 A~Z를 포함하고 있지 않으면서 그뒤에 ove가 놓이는 행을 일치시킨다.
\ (..\ )	일치된 문자를 기억한다.	S^(love\ ) able^ler/	지정되는 부분에 꼬리표를 붙이고 꼬리표번호1로서 그것을 기억한다. 후에 참조하기 위해 패턴참조로 \ 1을 사용한다. 패턴의 제일 왼쪽 부분의 첫 꼬리표를 가지고 시작하는데 바로 그 꼬리표를 사용할수 있다. 실례로 love는 등록기 1에 기억되고 교체문자렬에서 사용된다. lovable 은lover와 교체된다.
&	교체문자렬에서 사용할수 있도록 탐색문자렬을 기억한다.	S/love/**&**/	&는 탐색문자렬을 나타낸다. 문자렬 love는 별표안에 놓인 그자체로 교체된다. 즉 love는 **love**로 된다.
\ <	단어시작문자	/\ <love/	love로 시작하는 한개 단어를 포함하는 행을 일치시킨다.
\ >	단어끝문자	/love\ >/	Love로 끝나는 한개 단어를 포함하는 행을 일치시킨다.
x\ {m\ } x\ {m,\ } x\ {m,n\ }	문자 x의 m번 반복, 최소한 m번, 최소한 m번과 n번보다 많지 않게 반복	/0\ {5\ }/	행에 5개의 0이 있으면 적어도 5개의 령 또는 5~10개의 령을 대신한다.

제 6 절. sed실례

% cat datafile

northwest	NW	Charles Main	3.0	.98	3	34
western	WE	Sharon Gray	5.3	.97	5	23
southwest	SW	Lewis Dalsass	2.7	.8	2	18
southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Patricia Hemenway	4.0	.7	4	17
eastern	EA	TB Savage	4.4	.84	5	20
northeast	NE	AM Main Jr	5.1	.94	3	13
north	NO	Margot Weber	4.5	.89	5	9
central	CY	Ann Stephens	5.7	.94	5	13

1. 인쇄: p지령

실례 4-4

sed '/north/p' datafile

```
northwest   NW   Charles Main        3.0   98    3   34
northwest   NW   Charles Main        3.0   98    3   34
western     WE   Sharon Gray         5.3   .97   5   23
southwest   SW   Lewis Dalsass       2.7   .8    2   18
southern    SO   Suan Chin           5.1   .95   4   15
southeast   SE   Patricia Hemenway   4.0   .7    4   17
eastern     EA   TB Savage           4.4   .84   5   20
northeast   NE   AM Main Jr          5.1   .94   3   13
northeast   NE   AM Main Jr          5.1   .94   3   13
north       NO   Margot Weber        4.5   .89   5   9
north       NO   Margot Weber        4.5   .89   5   9
central     CT   Ann Stephens        5.7   .94   5   13
```

설명

기정값에 의해 표준출력으로 모든 행을 인쇄한다. 만일 패턴 north가 발견되면 sed는 다른 모든 행외에 그 행을 인쇄한다.

실례 4-5

sed –n '/north/p' datafile

```
northwest   NW   Chales          3.0   .98   3   34
northeast   NE   AM Main Jr.     5.1   .94   3   13
north       NO   Margot Weber    4.5   .89   5   9
```

설명

-n추가선택은 p지령과 함께 사용될 때 sed의 기정값에 의한 동작을 중지한다. 앞의 실례에서 보여 준바와 같이 -n추가선택이 없이 sed는 출력의 복사행들을 인쇄한다. 패턴 north를 포함하고 있는 행들만 –n을 사용할 때 인쇄된다.

% cat datafile

northwest	NW	Charles Main	3.0	.98	3	34
western	WE	Sharon Gray	5.3	.97	5	23
southwest	SW	Lewis Dalsass	2.7	.8	2	18
southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Patricia Hemenway	4.0	.7	4	17
eastern	EA	TB Savage	4.4	.84	5	20
northeast	NE	AM Main Jr	5.1	.94	3	13
North	NO	Margot Weber	4.5	.89	5	9
central	CY	Ann Stephens	5.7	.94	5	13

2. 지우기: d지령

실례 4-6

```
sed '3d' datafile
northwest   NW   Charles Main        3.0   98    3   34
western     WE   Sharon Gray         5.3   .97   5   23
southern    SO   Suan Chin           5.1   .95   4   15
southeast   SE   Patricia Hemenway   4.0   .7    4   17
eastern     EA   TB Savage           4.4   .84   5   20
northeast   NE   AM Main Jr          5.1   .94   3   13
north       NO   Margot Weber        4.5   .89   5   9
central     CTAnn Stephens           5.7   .94   5   13
```

설명

세번째 행을 삭제한다. 다른 모든 행은 기정값으로 현시장치에 인쇄된다.

실례 4-7

```
Sed '3,$d' datafile
northwest   NW   Charles Main   3.0   .98   3   34
western     WE   Sharon Gray    5.3   .97   5   23
```

설명

세번째 행에서부터 마지막행까지 지워 진다. 남아 있는 행은 인쇄된다. 화페문자

는 파일의 마지막행을 나타낸다. 반점을 범위(range)연산자라고 부른다. 이 실례에서 주소범위는 세번째 행에서 시작하여 $로 표시되는 마지막행에서 끝난다.

실례 4-8

sed '$d' datafile

northwest	*NW*	*Charles Main*	*3.0*	*98*	*3*	*34*
western	*WE*	*Sharon Gray*	*5.3*	*.97*	*5*	*23*
southwest	*SW*	*Lewis Dalsass*	*2.7*	*.8*	*2*	*18*
southern	*SO*	*Suan Chin*	*5.1*	*.95*	*4*	*15*
southeast	*SE*	*Patricia Hemenway*	*4.0*	*.7*	*4*	*17*
eastern	*EA*	*TB Savage*	*4.4*	*.84*	*5*	*20*
northeast	*NE*	*AM Main Jr*	*5.1*	*.94*	*3*	*13*
north	*NO*	*Margot Weber*	*4.5*	*.89*	*5*	*9*

설명

마지막행을 지운다. 화폐문자 $는 마지막행을 나타낸다. 기정값은 d지령에 의해 영향을 받는것을 제외한 모든 행들을 인쇄하는것이다.

실례 4-9

sed '/north/d' datafile

western	*WE*	*Sharon Gray*	*5.3*	*.97*	*5*	*23*
southwest	*SW*	*Lewis Dalsass*	*2.7*	*.8*	*2*	*18*
southern	*SO*	*Suan Chin*	*5.1*	*.95*	*4*	*15*
southeast	*SE*	*Patricia Hemenway*	*4.0*	*.7*	*4*	*17*
eastern	*EA*	*TB Savage*	*4.4*	*.84*	*5*	*20*
northeast	*NE*	*AM Main Jr*	*5.1*	*.94*	*3*	*13*
north	*NO*	*Margot Weber*	*4.5*	*.89*	*5*	*9*
central	*CT*	*Ann Stephens*	*5.7*	*.94*	*5*	*13*

설명

패턴 north를 포함하고 있는 모든 행이 지워 진다. 남아 있는 행들은 인쇄된다.

3. 교환: s지령

실례 4-10

sed 's/west/north/g' datafile

northnorth	*NW*	*Charles Main*	*3.0*	*98*	*3*	*34*
northern	*WE*	*Sharon Gray*	*5.3*	*.97*	*5*	*23*
southnorth	*SW*	*Lewis Dalsass*	*2.7*	*.8*	*2*	*18*
southern	*SO*	*Suan Chin*	*5.1*	*.95*	*4*	*15*
southeast	*SE*	*Patricia Hemenway*	*4.0*	*.7*	*4*	*17*

eastern EA TB Savage 4.4 .84 5 20

northeast NE AM Main Jr 5.1 .94 3 13
north NO Margot Weber 4.5 .89 5 9
central CT Ann Stephens 5.7 .94 5 13

설명

s지령은 교환기능을 가진다. 지령의 끝에 있는 g기발은 교환이 전체 행에 대하여 진행된다는것을 지적한다. 즉 여러개의 west가 존재하면 north로 교체된다. 만일 지령이 없으면 매개 행에서 처음 발생되는 한개 west만이 north로 교체된다.

실례 4-11

sed –n ‘s/^west/north/p’ datafile

northern WE Sharon Gray 5.3 .97 5 23

설명

s지령은 교환기능을 가진다. 지령의 끝에 있는 p기발을 가진 -n추가선택은 교환이 발생하는 그 행만을 인쇄하도록 sed에 알린다. 즉 west가 행의 시작에서 탐색되고 north와 교체되면 바로 그 행들을 인쇄한다.

% cat datafile

northwest	NW	Charles Main	3.0	.98	3	34
western	WE	Sharon Gray	5.3	.97	5	23
southwest	SW	Lewis Dalsass	2.7	.8	2	18
southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Patricia Hemenway	4.0	.7	4	17
eastern	EA	TB Savage	4.4	.84	5	20
northeast	NE	AM Main Jr	5.1	.94	3	13
North	NO	Margot Weber	4.5	.89	5	9
central	CY	Ann Stephens	5.7	.94	5	13

실례 4-12

sed -s/[0-9][0-9]$/&.5/- datafile

northwest NW Charles Main 3.0 98 3 34
western WE Sharon Gray 5.3 .97 5 23
southwest SW Lewis Dalsass 2.7 .8 2 18
southern SO Suan Chin 5.1 .95 4 15
southeast SE Patricia Hemenway 4.0 .7 4 17
eastern EA TB Savage 4.4 .84 5 20

northeast	*NE*	*AM Main Jr*	*5.1*	*.94*	*3*	*13*
north	*NO*	*Margot Weber*	*4.5*	*.89*	*5*	*9*
central	*CT*	*Ann Stephens*	*5.7*	*.94*	*5*	*13*

설명

교체문자렬에서 문자 &[3]는 탐색문자렬에서 발견된것을 정확히 표시한다. 2개의 수자로 끌나는 매 행이 그자체에 의해 교체되고 5가 추가된다.

실례 4-13

sed –n 's/Hemenway/Jones/gp' datafile

southeast	*SE*	*Paricia Jones*	*4.0*	*.7*	*4*	*17*

설명

발생되는 모든 Hemenway이 Johns로 교체되며 변화되는 행만이 인쇄된다. p지령과 조합되는 -n추가선택은 기정값의 출력을 정지시킨다. g는 행에서 전체 치환을 의미한다.

실례 4-14

sed –n 's/(Mar\ )got/\ lianne/p' datdfile

north	*NO*	*Marianne Weber*	*4.5*	*.89*	*5*	*9*

설명

패턴 Mar는 괄호안에 놓이며 특정의 등록기에 꼬리표 1로서 기억된다. 이것은 \ 1로서 교체문자렬에서 참조될수 있다. 그다음 Margot는 marianne으로 교체된다.

실례 4-15

sed 's#3#88#g' datafile

northwest	*NW*	*Charles Main*	*88.0*	*.98*	*88*	*884*
western	*WE*	*Sharon Gray*	*5.88*	*.97*	*5*	*288*
southwest	*SW*	*Lewis Dalsass*	*2.7*	*.8*	*2*	*18*
southern	*SO*	*Suan Chin*	*5.1*	*.95*	*4*	*15*
southeast	*SE*	*Patricia Hemenway*	*4.0*	*.7*	*4*	*17*
eastern	*EA*	*TB Savage*	*4.4*	*.84*	*5*	*20*
northeast	*NE*	*AM Main Jr.*	*5.1*	*.94*	*88*	*188*
north	*NO*	*Margot Weber*	*4.5*	*.89*	*5*	*9*
central	*CT*	*Ann Stephens*	*5.7*	*.94*	*5*	*188*

설명

s지령 다음의 문자는 탐색문자렬과 교체문자렬사이의 분리기이다. 분리기문자는 기정값으로는 빗선문자이지만 변화될수 있다(s지령이 사용될 때만). 문자가 s지령

3. 교체문자렬에서 기호 &를 표시하기 위해서는 그것이 탈퇴되여야 한다. 즉 \ &를 사용하여야 한다.

의 뒤에 놓이는것은 새로운 문자렬분리기이다. 이 기술은 경로이름 또는 날자와 같은 한개 빗선문자를 포함하고 있는 패턴을 탐색할 때 유용하다.

```
% cat datafile
northwest    NW    Charles Main         3.0   .98   3   34
western      WE    Sharon Gray          5.3   .97   5   23
southwest    SW    Lewis Dalsass        2.7   .8    2   18
southern     SO    Suan Chin            5.1   .95   4   15
southeast    SE    Patricia Hemenway    4.0   .7    4   17
eastern      EA    TB Savage            4.4   .84   5   20
northeast    NE    AM Main Jr           5.1   .94   3   13
North        NO    Margot Weber         4.5   .89   5    9
central      CT    Ann Stephens         5.7   .94   5   13
```

4. 선택된 행의 범위: 반점

실례 4-16

```
sed -n '/west/,/east/p' datafile
northwest    NW    Charles Main         3.0   .98   3   34
western      WE    Sharon Gray          5.3   .97   5   23
southwest    SW    Lewis Dalsass        2.7   .8    2   18
southern     SO    Suan Chin            5.1   .95   4   15
southeast    SE    Patricia Hemenway    4.0   .7    4   17
```

설명

west와 east사이의 패턴범위에 있는 모든 행을 인쇄한다. 만일 west가 east 다음의 행에 놓이면 west로부터 다음 east까지 또는 처음에 오는 파일의 끝까지의 행들이 인쇄된다. 방향는 범위를 지적한다.

실례 4-17

```
sed -n  '5,/^northeast/p' datafile
southeast    SE    Patricia Hemenway    4.0   .7    4   17
eastern      EA    TB Savage            4.4   .84   5   20
northeast    NE    AM Main Jr.          5.1   .94   3   13
```

설명

5행부터 northeast로 시작되는 첫번째 행까지의 행들을 인쇄한다.

실례 4-18

```
sed 'west/,/east/s/$/**VACA**/' datafile
```

```
northwest   NW   Charles Main        3.0   98    3   34**VACA**
western     WE   Sharon Gray         5.3   .97   5   23**VACA**
southwest   SW   Lewis Dalsass       2.7   .8    2   18**VACA**
southern    SO   Suan Chin           5.1   .95   4   15**VACA**
southeast   SE   Patricia Hemenway   4.0   .7    4   17**VACA**
eastern     EA   TB Savage           4.4   .84   5   20
northeast   NE   AM Main Jr          5.1   .94   3   13
north       NO   Margot Weber        4.5   .89   5   9
central     CT   Ann Stephens        5.7   .94   5   13
```

설명

패턴 east와 west사이의 범위에 있는 행들에 대하여 행의 끝($)이 문자렬 **VACA**로 교체된다. 행바꾸기문자는 새로운 문자렬의 끝으로 이동된다. 방향는 범위를 표시한다.

5. 다중편집: e지령

실례 4-19

```
sed -e '1,3d' -e 's/Hemeway/Jones/ ' datafile
southern    SO   Suan Chin           5.1   .95   4   15
southeast   SE   Patricia Hemenway   4.0   .7    4   17
eastern     EA   TB Savage           4.4   .84   5   20
northeast   NE   AM Main             5.1   .94   3   13
north       NO   Margot Weber        4.5   .89   5   9
central     CT   Ann Stephens        5.7   .94   5   13
```

설명

-e추가선택은 다중편집을 할수 있도록 허용한다. 첫 편집은 1행부터 3행까지를 지운다. 두번째 편집은 Jones를 Hemeway로 바꾼다. 두 편집이 행별로 수행되기 때문에(즉 두 지령은 패턴공간의 현재행에서 실행된다.) 편집순서는 결과에 영향을 미치게 된다. 실례로 두 지령을 행에서 교체하였다면 첫 교체는 두번째 교체에 영향을 주게 된다.

```
% cat datafile
northwest   NW   Charles Main        3.0   .98   3   34
western     WE   Sharon Gray         5.3   .97   5   23
southwest   SW   Lewis Dalsass       2.7   .8    2   18
southern    SO   Suan Chin           5.1   .95   4   15
southeast   SE   Patricia Hemenway   4.0   .7    4   17
eastern     EA   TB Savage           4.4   .84   5   20
northeast   NE   AM Main Jr          5.1   .94   3   13
North       NO   Margot Weber        4.5   .89   5    9
central     CY   Ann Stephens        5.7   .94   5   13
```

6. 파일로부터 읽기: r지령

실례 4-20

```
% cat newfile
            ***SUAN HAS LEFT THE COMPANY ***
% sed '/Suan/r newfile' datafile
northwest   NW   Charles Main       3.0   98    3   34
western     WE   Sharon Gray        5.3   .97   5   23
southwest   SW   Lewis Dalsass      2.7   .8    2   18
southern    SO   Suan Chin          5.1   .95   4   15
            ***SUAN HAS LEFT THE COMPANY***
southeast   SE   Patricia Hemenway  4.0   .7    4   17
eastern     EA   TB Savage          4.4   .84   5   20
northeast   NE   AM Main            5.1   .94   3   13
north       NO   Margot Weber       4.5   .89   5   9
central     CT   Ann Stephens       5.7   .94   5   13
```

설명

r지령은 지정된 행을 파일로부터 읽는다. newfile의 내용은 패턴 Suan이 일치되는 행다음의 입력파일 datafile에 써넣어 진다. suan이 여러 행에서 출현할 때 newfile의 내용은 그것이 발생할 때마다 읽어 진다.

7. 파일에로 쓰기: w 지령

실례 4-21

```
sed -n '/north/w newfile' datafile
cat newfile
northwest   NW   Charles Main       3.0   98    3   34
northeast   NE   AM Main Jr         5.1   .94   3   13
north       NO   Margot Weber       4.5   .89   5   9
```

설명

w지령은 파일에로 지정된 행을 써넣는다. 패턴 north를 포함하는 모든 행은 newfile이라고 하는 파일에 써넣는다.

8. 추가: a 지령

실례 4-22

```
sed '/^north   /a\ \
--->THE NORTH SALES DISTRICT HAS MOVED<---' datafile
```

northwest	*NW*	*Charles Main*	*3.0*	*98*	*3*	*34*
western	*WE*	*Sharon Gray*	*5.3*	*.97*	*5*	*23*
southwest	*SW*	*Lewis Dalsass*	*2.7*	*.8*	*2*	*18*
southern	*SO*	*Suan Chin*	*5.1*	*.95*	*4*	*15*
southeast	*SE*	*Patricia Hemenway*	*4.0*	*.7*	*4*	*17*
eastern	*EA*	*TB Savage*	*4.4*	*.84*	*5*	*20*
northeast	*NE*	*AM Main Jr*	*5.1*	*.94*	*3*	*13*
north	*NO*	*Margot Weber*	*4.5*	*.89*	*5*	*9*
--->THE NORTH SALES DISTRICT HAS MOVED<---						
central	*CT*	*Ann Stephens*	*5.7*	*.94*	*5*	*13*

설명

a지령은 추가지령이다. 문자렬 --->THE NORTH SALES DISTRICT HAS MOVED<---은 north 다음에 공백이 놓일 때 패턴 *north*로 시작하는 행다음에 추가된다. 추가되는 본문은 추가지령다음의 행에 존재하여야 한다.

sed는 a지령다음에 한개 거꿀빗선을 요구한다. 행바꾸기문자[4]를 탈퇴시키기 위해 C쉘은 두번째 거꿀빗선을 리용하여 닫는 인용부호가 다음행에 놓이게 한다. 여러개의 행이 추가되면 마지막 하나의 행을 제외하고 모든 행은 거꿀빗선으로 끝나야 한다.

% cat datafile

northwest	NW	Charles Main	3.0	.98	3	34
western	WE	Sharon Gray	5.3	.97	5	23
southwest	SW	Lewis Dalsass	2.7	.8	2	18
southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Patricia Hemenway	4.0	.7	4	17
eastern	EA	TB Savage	4.4	.84	5	20
northeast	NE	AM Main Jr	5.1	.94	3	13
North	NO	Margot Weber	4.5	.89	5	9
central	CY	Ann Stephens	5.7	.94	5	13

9. 삽입: i 지령

실례 4-23

```
sed '/eastern/i\ \
MEW ENGLAND REGIONS \ \
```

[4] Bourne과 Korn쉘들은 행바꾸기를 탈퇴시키기 위해 두번째 거꿀빗선을 요구하지 않는다. 왜냐하면 이 쉘들에서는 인용부호가 같은 행에서는 일치되지 않아도 전체적으로 일치되면 되기때문이다.

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -┘ datafile
northwest    NW    Charles Main          3.0    98     3    34
western      WE    Sharon Gray           5.3    .97    5    23
southwest    SW    Lewis Dalsass         2.7    .8     2    18
southern     SO    Suan Chin             5.1    .95    4    15
southeast    SE    Patricia Hemenway     4.0    .7     4    17
NEW ENGLAND REGION
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
eastern      EA    TB Savage             4.4    .84    5    20
northeast    NE    AM Main Jr            5.1    .94    3    13
north        NO    Margot Weber          4.5    .89    5    9
central      CT    Ann Stephens          5.7    .94    5    13
```

설명

i지령은 삽입지령이다. 만일 패턴 eastern이 일치되면 i지령은 거꿀빗선다음에 오는 본문이 eastern을 포함하고 있는 행우에 삽입되게 한다. 거꿀빗선은 제일 마지막행 하나를 제외하고 삽입되는 매 행의 뒤에 놓여야 한다(나머지의 거꿀빗선은 C쉘을 위한것이다.).

10. 다음: n 지령

실례 4-24

```
sed '/eastern/{ n; s/AM/Archie /; }' datafile
northwest    NW    Charles Main          3.0    98     3    34
western      WE    Sharon Gray           5.3    .97    5    23
southwest    SW    Lewis Dalsass         2.7    .8     2    18
southern     SO    Suan Chin             5.1    .95    4    15
southeast    SE    Patricia Hemenway     4.0    .7     4    17
eastern      EA    TB Savage             4.4    .84    5    20
→ northeast  NE    AM Main Jr            5.1    .94    3    13
north        NO    Margot Weber          4.5    .89    5    9
central      CT    Ann Stephens          5.7    .94    5    13
```

설명

패턴 eastern이 한 행에서 일치되면 n지령은 sed가 다음입력행(AM Main Jr를 가진 행)을 받아서 그것으로 패턴을 교체하고 AM으로 Archie를 교체하며 그 행을 인쇄한 다음 동작이 계속되도록 한다.

11. 변경: y 지령

실례 4-25

```
sed '1,3y/abcde£ghijklmnopqrstuvwxyz/ABCDEFGHXJKL
```

```
MMOPQRSTUVWXYZ/'    datafile

NORTHWEST     NW   CHARLES MAIN        3.0   98    3   34
WESTERN       WE   SHARON GRAY         5.3   .97   5   23
SOUTHWEST     SW   LEWIS DALSASS       2.7   .8    2   18
southern      SO   Suan Chin           5.1   .95   4   15
southeast     SE   Patricia Hemenway   4.0   .7    4   17
eastern       EA   TB Savage           4.4   .84   5   20
northeast     NE   AM Main Jr          5.1   .94   3   13
north         NO   Margot Weber        4.5   .89   5   9
central       CT   Ann Stephens        5.7   .94   5   13
```

설명

y지령은 행 1부터 3까지의 행들에 대해 모든 소문자를 대문자로 변경시킨다. 정규표현메타문자는 이 지령을 사용할수 없다.

```
% cat datafile
northwest     NW   Charles Main        3.0   .98   3   34
western       WE   Sharon Gray         5.3   .97   5   23
southwest     SW   Lewis Dalsass       2.7   .8    2   18
southern      SO   Suan Chin           5.1   .95   4   15
southeast     SE   Patricia Hemenway   4.0   .7    4   17
eastern       EA   TB Savage           4.4   .84   5   20
northeast     NE   AM Main Jr          5.1   .94   3   13
north         NO   Margot Weber        4.5   .89   5    9
central       CY   Ann Stephens        5.7   .94   5   13
```

12. 중지: q지령

실례 4-26

```
sed -5q' datafile
northwest   NW   Charles Main        3.0   98    3   34
western     WE   Sharon Gray         5.3   .97   5   23
southwest   SW   Lewis Dalsass       2.7   .8    2   18
southern    SO   Suan Chin           5.1   .95   4   15
southeast   SE   Patricia Hemenway   4.0   .7    4   17
```

설명

q지령은 다섯번째 행을 인쇄한후 탈퇴하도록 sed프로그람이 중지되게 한다.

실례 4-27

sed '/Lewis/{ s / Lewis/Joseph / ; q ; }' datafile

northwest	*NW*	*Charles Main*	*3.0*	*98*	*3*	*34*
western	*WE*	*Sharon Gray*	*5.3*	*.97*	*5*	*23*
southwest	*SW*	*Lewis Dalsass*	*2.7*	*.8*	*2*	*18*

설명

패턴 Lewis가 행에서 일치될 때 우선 교환지령(s)은 Lewis를 Joseph로 교체하며 그다음 q지령은 sed프로그람이 탈퇴하도록 한다.

13. 보존(Holding)과 얻기(Getting): h와 g지령

실례 4-28

sed -e '/northeast/h' -e '$G' datafile

	northwest	*NW*	*Charles Main*	*3.0*	*98*	*3*	*34*
	western	*WE*	*Sharon Gray*	*5.3*	*.97*	*5*	*23*
	southwest	*SW*	*Lewis Dalsass*	*2.7*	*.8*	*2*	*18*
	southern	*SO*	*Suan Chin*	*5.1*	*.95*	*4*	*15*
	southeast	*SE*	*Patricia Hemenway*	*4.0*	*.7*	*4*	*17*
	eastern	*EA*	*TB Savage*	*4.4*	*.84*	*5*	*20*
→	*northeast*	*NE*	*AM Main Jr*	*5.1*	*.94*	*3*	*13*
	north	*NO*	*Margot Weber*	*4.5*	*.89*	*5*	*9*
	central	*CT*	*Ann Stephens*	*5.7*	*.94*	*5*	*13*
→	*northeast*	*NE*	*AM Main Jr*	*5.1*	*.94*	*3*	*13*

설명

sed가 파일을 처리하므로 매 행은 패턴공간이라고 부르는 림시적인 완충기에 기억된다. 행이 출력될 때 지워 지지 않거나 인쇄될수 있으면 그것이 처리된후에 현시장치에 인쇄된다. 그다음에 패턴공간은 지워 지고 다음의 입력행은 처리를 위해 기억된다. 이 실례에서는 패턴 northeat를 포함하고 있는 행을 탐색한 다음 그 행을 패턴공간에 배치하며 h지령은 그것을 복사하여 보존완충기(holding buffer)라고 하는 다른 특수한 완충기에 기억한다.

두번째 sed지령에서 마지막행($)이 나타나면 g지령은 sed가 보존완충기로부터 행을 읽어서 패턴공간완충기에 다시 넣어 현재 거기에(이 경우에 마지막행에) 기억된 행에 첨가되게 한다. 간단하게 설명하면 패턴 northeast를 가진 임의의 행은 복사되여 파일의 끝에 덧붙여 진다(그림 4-1).

그림 4-1. 패턴공간과 보존완충기(실례 4-31)

% cat datafile

northwest	NW	Charles Main	3.0	.98	3	34
western	WE	Sharon Gray	5.3	.97	5	23
southwest	SW	Lewis Dalsass	2.7	.8	2	18
southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Patricia Hemenway	4.0	.7	4	17
eastern	EA	TB Savage	4.4	.84	5	20
northeast	NE	AM Main Jr	5.1	.94	3	13
north	NO	Margot Weber	4.5	.89	5	9
central	CY	Ann Stephens	5.7	.94	5	13

실례 4-29

northwest	*NW*	*Charles Main*	*3.0*	*98*	*3*	*34*
southwest	*SW*	*Lewis Dalsass*	*2.7*	*.8*	*2*	*18*
southern	*SO*	*Suan Chin*	*5.1*	*.95*	*4*	*15*
southeast	*SE*	*Patricia Hemenway*	*4.0*	*.7*	*4*	*17*
eastern	*EA*	*TB Savage*	*4.4*	*.84*	*5*	*20*
northeast	*NE*	*AM Main Jr*	*5.1*	*.94*	*3*	*13*

north	*NO*	*Margot Weber*	*4.5*	*.89*	*5*	*9*
central	*CT*	*Ann Stephens*	*5.7*	*.94*	*5*	*13*
→ *western*	*WE*	*Sharon Gray*	*5.3*	*.97*	*5*	*23*

설명

한 행에서 패턴 WE를 탐색하면 h지령은 그 행이 패턴공간으로부터 보존완충기에 복사되게 한다. 행이 보존완충기에 기억되면 그것을 후에 리용할수 있다(g혹은 h지령). 이 실례에서는 패턴 WE가 탐색되면 그것이 탐색된 행이 먼저 패턴완충기에 기억한다. 그다음에 h지령은 보존완충기에 행의 사본을 넣는다. D지령은 패턴완충기에서 행의 복사를 삭제한다. 두번째 지령은 행에서 CT를 탐색하며 탐색되면 sed는 그것을 보존완충기에 기억된 행을 읽어(G) 현재패턴공간의 행에 그것을 덧붙인다. 간단히 말하여 WE를 포함하고 있는 행은 이동되여 CT를 포함하고 있는 행의 뒤에 첨가된다(84페지의《보존과 교환: h와 x지령》을 참고).

실례 4-30

Sed –e '/northeast/h'   -e '$g' datafile

northwest	*NW*	*Charles Main*	*3.0*	*98*	*3*	*34*
western	*WE*	*Sharon Gray*	*5.3*	*.97*	*5*	*23*
southwest	*SW*	*Lewis Dalsass*	*2.7*	*.8*	*2*	*18*
southern	*SO*	*Suan Chin*	*5.1*	*.95*	*4*	*15*
southeast	*SE*	*Patricia Hemenway*	*4.0*	*.7*	*4*	*17*
eastern	*EA*	*TB Savage*	*4.4*	*.84*	*5*	*20*
→ *northeast*	*NE*	*AM Main Jr*	*5.1*	*.94*	*3*	*13*
north	*NO*	*Margot Weber*	*4.5*	*.89*	*5*	*9*
→ *northeast*	*NE*	*AM Main Jr*	*5.1*	*.94*	*3*	*13*

설명

sed가 파일을 처리할 때 매 행은 패턴공간이라고 부르는 림시완충기에 기억된다. 그 행이 출력에서 지워 지거나 인쇄될수 있으면 그 행은 처리된후에 현시장치에 표시된다. 그다음에 패턴공간은 지워 지고 다음 입력행이 처리를 위해 거기에 기억된다. 이 실례에서는 패턴 northeast를 포함하고 있는 행이 발견된후 그 행이 패턴공간에 넣어 진다. h지령은 그행을 복사하여 보존완충기라는 다른 특수한 완충기에 넣는다. 두번째 sed지령에서 마지막행이 나타나면 g지령은 sed가 보존완충기로부터 행을 읽어서 패턴공간에 다시 넣어 거기에(이 경우에는 마지막행) 현재 기억되여 있는 행을 교체하게 한다. 간단히 설명하면 패턴 northeast을 포함하고 있는 행은 복사되고 이동되여 파일의 마지막행을 덧쓰기한다.

% cat datafile

northwest	NW	Charles Main	3.0	.98	3	34
western	WE	Sharon Gray	5.3	.97	5	23

southwest	SW	Lewis Dalsass	2.7	.8	2	18
southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Patricia Hemenway	4.0	.7	4	17
eastern	EA	TB Savage	4.4	.84	5	20
northeast	NE	AM Main Jr	5.1	.94	3	13
north	NO	Margot Weber	4.5	.89	5	9
central	CY	Ann Stephens	5.7	.94	5	13

실례 4-31

sed -e '/WE/{h; d; }' -e '/CT/{g; }'   datafile

```
northwest   NW   Charles Main        3.0   98    3   34
southwest   SW   Lewis Dalsass       2.7   .8    2   18
southern    SO   Suan Chin           5.1   .95   4   15
southeast   SE   Patricia Hemenway   4.0   .7    4   17
eastern     EA   TB Savage           4.4   .84   5   20
northeast   NE   AM Main Jr          5.1   .94   3   13
north       NO   Margot Weber        4.5   .89   5   9
western     WE   Sharon Gray         53    .97   5   23
```

설명

패턴 WE가 발견되면 h지령은 보존완충기에 그 행을 복사하고 d지령은 패턴공간에서 그 행을 삭제한다. 패턴 CT가 발견되면 g지령은 보존완충기에 있는 사본을 읽어 현재패턴공간에 있는 행에 덧쓰기한다. 간단히 설명하여 패턴 WE가 있는 행을 CT를 가지고 있는 행우에 덧쓰기 위해 이동한다(그림 4-1).

14. 보존과 교환: h 와 x 지령

실례 4-32

sed -e '/Patricia/h' -e '/Margot/x' datafile

```
northwest   NW   Charles Main        3.0   98    3   34
western     WE   Sharon Gray         5.3   .97   5   23
southwest   SW   Lewis Dalsass       2.7   .8    2   18
southern    SO   Suan Chin           5.1   .95   4   15
southeast   SE   Patricia Hemenway   4.0   .7    4   17
eastern     EA   TB Savage           4.4   .84   5   20
northeast   NE   AM Main Jr          5.1   .94   3   13
southeast   SE   Patricia Hemenway   4.0   .7    4   17
```

central CT Ann Stephens 5.7 .94 5 13

설명

x지령은 현재패턴공간과 보존완충기의 내용을 교체한다. 패턴 patricia를 가진 행이 발견되면 그 행을 보존완충기에 기억시킨다. Margot가 있는 행이 발견되면 패턴공간과 보존완충기에 있는 행을 교체한다. 간단히 설명하면 Margot를 가지고 있는 행을 partricia를 가지고 있는 행과 교체한다.

제 7 절. sed의 스크립트작성법

sed스크립트는 파일에 있는 sed지령들의 목록이다. 지령들이 파일에 있다는것을 sed가 인식하도록 하기 위해 지령행에서 sed를 호출할 때 sed스크립트의 이름앞에 –f추가선택을 리용하면 된다. sed는 아주 독특한 방법으로 스크립트에 행들을 입력한다. 지령끝에 임의의 공백이나 본문이 있을수 있다. 만일 지령들이 그자체가 어떤 행에도 놓이지 않으면 그 지령들은 반두점으로 끝나야 한다. 입력파일로부터 한행을 읽어 패턴완충기에 복사하고 sed스크립트에 있는 모든 지령들을 그 행에서 실행한다. 이 행이 처리된 후에 입력파일에서 다음의 행을 패턴완충기에 써넣고 스크립트에 있는 모든 지령들을 그 행에서 실행한다. sed는 문법이 틀리면 정확히 동작하지 못한다.

sed스크립트의 우점은 사용자가 지령행에서와 같이 쉘과의 대화에 대하여 주의를 돌리지 않아도 된다는것이다. sed지령들이 쉘에 의해 해석되지 않도록 하기 위해 인용부호를 사용할 필요는 없다. 사실 사용자는 sed스크립트의 인용부호들이 탐색패턴부분이 아니면 리용할수 없다.

% cat datafile

Northwest	NW	Charles Main	3.0	.98	3	34
Western	WE	Sharon Gray	5.3	.97	5	23
Southwest	SW	Lewis Dalsass	2.7	.8	2	18
Southern	SO	Suan Chin	5.1	.95	4	15
Southeast	SE	Patricia Hemenway	4.0	.7	4	17
Eastern	EA	TB Savage	4.4	.84	5	20
Northeast	NE	AM Main Jr	5.1	.94	3	13
North	NO	Margot Weber	4.5	.89	5	9
Central	CY	Ann Stephens	5.7	.94	5	13

1. sed 스크립트의 실례

실례 4-33

% cat sedding1 *(Look at the contents of the sed script.)*

1. *# My first sed script by Jack Sprat*

```
2.   /Lewis/a\
3 .          Lewis is the TOP Salesperson for April!!\

             Lewis is moving to the southern district next month. \

4            CONGRATULATIONS!
5.   /Margot/c\
      **************************
      MARGOT HAS RETIRED\
      **************************

6.   li\
     EMPLOYEE DATABASE  \
          ------------------------------
7.   $d
```

% sed -f sedding1datafile *(Execute the sed script commands; the input file is datafile.)*

```
EMPLOYEE DATABASE
- - - - - - - - - - - - - - - - - - - - - - - - - -
northwest   NW    Charles Main        3.0   98    3   34
western     WE    Sharon Gray         5.3   .97   5   23
southwest   SW    Lewis Dalsass       2.7   .8    2   18
Lewis is the TOP Salesperson for April!!
Lewis is moving to the southern district next month
CONGRATULATIONS !
southern    SO    Suan Chin           5.1   .95   4   15
southeast   SE    Patricia Hemenway   4.0   .7    4   17
eastern     EA    TB Savage           4.4   .84   5   20
northeast   NE    AM Main Jr          5.1   .94   3   13

******************************
 MARGOT HAS RETIRED
***************************
```

설명

1. 이 행은 해설이다. 해설은 그자체가 행에 존재하여야 하며 문자 #로 시작하여야 한다.
2. 행이 패턴 Lewis를 가지고 있으면 그다음의 3개의 행들이 그 행에 추가된다.
3. 마지막행을 제외하고 추가되는 매행은 거꿀빗선으로 끝난다. 거꿀빗선뒤에는 행바꾸기문자가 놓여야 한다. 임의의 행바꾸기문자뒤에 어떤 본문, 지어는 한개의 공백이 있어도 오유통보문을 내보낸다.
4. 추가되는 마지막행에는 거꿀빗선(\ )이 없다. 그것은 sed에게 이것이 추가되는 마지막행이며 다음행이 또 다른 지령이라는것을 알려 준다.
5. 패턴 Margot를 포함하고 있는 임의의 행들을 본문의 그다음 3개의 행으로 교체한다(C지령).

6. 다음의 2개의 행은 행 1에 삽입(I지령)된다.
7. 마지막행($)은 지워 진다.

실례 4-34

% **cat sedding2** *(Look at the contents of the sed script.)*
# This script demonstrates the use of curly braces to nest *addresses*
# *and commands. Comments are* preceded by a *pound* sign (#) and must
# be on a line by themselves. *Commands are* terminated, with a newline
# or semicolon.   If there is any text after a *command, even one*
# space, *you receive an error* message:
#   **sed: Extra text *at end of command;***

```
1.  western/, /southeast/{
        /" *$/d
            /Suan/{ h; d; }
          }
2.  /Ann/g
3.  s/TB \ (Savage\ )/Thomas \ 1/
```

% **sed –f sedding2 datafile**

northwest	*NW*	*Charles Main*	*3.0*	*98*	*3*	*34*
western	*WE*	*Sharon Gray*	*5.3*	*.97*	*5*	*23*
southwest	*SW*	*Lewis Dalsass*	*2.7*	*.8*	*2*	*18*
southeast	*SE*	*Patricia Hemenway*	*4.0*	*.7*	*4*	*17*
eastern	*EA*	*TB Savage*	*4.4*	*.84*	*5*	*20*
northeast	*NE*	*AM Main Jr*	*5.1*	*.94*	*3*	*13*
north	*NO*	*Margot Weber*	*4.5*	*.89*	*5*	*9*
southern	*SO*	*Suan Chin*	*5.1*	*.95*	*4*	*15*

설명

1. western으로 시작되고 southeast로 끝나는 행범위에서 공백행들은 삭제된다. 그다음 suan과 일치되는 행들은 패턴완충기에서 보존완충기에로 복사된 다음 패턴완충기는 완충기로부터 지우기된다.
2. g지령은 패턴 Ann이 일치되면 보존완충기에 있는 행을 패턴완충기에 복사하여 그우에 덧쓰기한다.
3. 패턴 TB Savage를 가지는 모든 행은 Thomas와 꼬리가 붙여 진 패턴 즉 Savage로 교체된다. 탐색문자렬에서 Savage를 다시 사용할수 있도록 문자렬에 꼬리표를 붙여 탈퇴된 괄호안에 놓는다. 이것은 꼬리번호가 1이며 \ 1로서 참조된다.

2. 개괄

표 4-4에 sed지령들과 그 기능들을 보여 주었다.

표 4-4. sed 개괄

지령	무엇을 수행하는가
sed-n'/sentimental/p' file	sentimental을 포함하고 있는 모든 행을 현시장치에 출력한다. 파일 filex는 변화되지 않는다. -n 추가선택이 없이 sentimental을 가진 모든 행들이 중복되여 인쇄된다.
sed'1,3d'filex>newfilex	filex의 행 1, 2, 3이 지워 지며 newfilex에서 변화된 값이 인쇄된다.
sed'/[Dd]amiel/d' filex	Daniel이나 daniel를 포함하고 있는 행들을 인쇄한다.
sed-n' 15,20p' filex	행 15부터 20까지를 인쇄한다.
sed'1,10s/Montana/MT/g' filex	행 1부터 10까지에서 Montana를 MT로 교체한다.
sed'/March^!d'filex (csh) sed'/March/!d' filex (sh)	March를 포함하지 않는 모든 행을 지운다. (거꿀빗선은 리력문자를 탈퇴시키기 위해 csh에서만 리용된다.).
sed'/report/s/5/8/' filex	report를 포함하고 있는 모든 행에서 처음으로 발생하는 5~8개의 문자들을 변화시킨다.
sed's/...//' filex	매행에서 첫 4개 문자를 지운다.
sed's$//' filex	매행에서 마지막 3개 문자를 지운다.
sed'/east/,/west/s/North/South/'filex	east로부터 west까지의 범위에 있는 모든 행에서 North를 South로 교체한다.
sed-n'/Time off/wtimefile' filex	파일 Timefile에 Time off를 포함하는 모든 행을 써넣는다.
sed's∧([Oo]ccur\)ence∧1rence/' file	Occurrence 이나 occurrence 을 사용하여 Occurrence이나 occurrence중의 하나를 교체한다.
sed -n'l' filex	\ nn형식으로 문자를 인쇄하지 않으며 >로서 타브를 표시하는 모든 행을 인쇄한다. 여기서 nn은 문자의 8진수값이다.

UNIX도구들에 대한 련습

련습문제 2: sed련습

steve Blenheim:238-923-7366:95 Latham lane,Easton,PA 83755:11/122/56:20300
Betty Boop:245-836-8395:635 Custesy Lane,Hollywood,CAA 91464:6/23/23:14500
Igor Chevsky:385-375-8395:3567 Populus Place,Caldwell,Nj 23875:6/18/68:23400

Norma Corder:397-857-2735:74 Pine Street,Dearborn,MI23874:3/28/45:245700
JenniferCowan:548-834-2348:583LaurelAve.,Kingsville,TX 83745:10/1/35:58900
Jon DeLoach:408-253-3122-123 Park St.,San Jose,CA 04086:7/25/53:85100
Karen Evich:284-758-2857:23 Edgecliff Place,Lincoln,NB 92743:7/25/53:85100
Karen Evich:284-758-2857:23 Edgecliff Place,Lincoln,NB 92743:11/3/35:58200
Karen Evich:284-758-2857:23 Edgecliff Place,Lincoln,NB 92743:11/3/35:58200
Fred Fardbarkle:674-843-1385:20 Parak Lane,DeLuth,MN 23850:4/12/23:780900
Fred Fardbarkle:674-843-1385:20 Parak Lane,DeLuth,MN 23850:4/12/23:780900
Lori Gortz:327-832—5728:3465 Mirlo Street,Peabody,MA 34756:10/2/65:35200
Paco Gutierrez:835-365-1284:454 Easy Street,Decatur,IL 75732:2/28/53:123500
Ephram Hardy:293-259-5395:235 CarltonLane,Joliet,IL 73858:8/12/20:56700
James Ikeda:834-938-8376:23445 Aster Ave.,Allentown,NJ 83745:12/1/38:45000
Barbara Kertz:385-573-8326:832 Ponce Drive,Gary,IN 8375612/1/46/268500
Lesley Kirstin:408-456-1234:4 Hardvard Square,Boston,MA 02133:4/22/62:52600
William Kopf:846-836-2837:6937 Ware Road,Milton,PA 93756:9/21/46:43500
Sir Lancelot:837-835-8257:474 Camelot Boulevard,Bath,WY 28356:5/13/69:24500
Jesse Neal:408-233-8971:45 Rose Terrace, San Francisco, CA 92303:2/3/36:25000
Zippy Pinhead:834-823-8319:2356 Bisarro Ave.,Farmount, IL 84357:1/1/67:89500
Arthur Putie:923-835-8745:23 Wimp Lane,Kensington,DL 38758:8/31/69:126000
PopeyeSailor:156-454-3322:945Bluto Street,Anywhere,USA 29358:3/19/35:22350
Jose Santiago:385-898-8357:38 Fife Way, Abilene,TX 39673:1/5/58:95600
TommySavage:408-724-0140:1222OxbowCourt,Sunnyvale,CA 087:5/19/66:34200
Yukio Takeshida:387-827-1095:13 Uno Lane,Ashville,NC 23556:7/1/29:57000
Vinh Tranh:438-910-7449:8235 Maple Street,Wilmington,VM 29085:9/23/63:68900

(CD에 있는 databook라고 하는 자료기지를 참조하시오.)

1. Jon의 이름을 Jonathan으로 교체하시오.
2. 첫 3개의 행을 지우시오.
3. 다섯번째 행부터 열번째 행까지 인쇄하시오.
4. Lane이 들어 있는 행을 지우시오.
5. 생일이 11월과 12월인 행들을 모두 지우시오.
6. 3개의 별표식을 Find로 시작하는 행의 끝에 첨부하시오.
7. Jose를 가지고 있는 행을 Jose HAS RETIRED로 교체하시오.
8. Popeye의 생일을 11/14/46으로 바꾸시오.
9. 빈 행을 모두 삭제하시오.
10. 다음의것을 Sed스크립트에 써넣으시오.

    ㄱ. 첫행앞에 제목 RERSOMNEL FILE을 삽입하시오.

    ㄴ. 500원까지의 월급을 지우시오.

    ㄷ. 첫 이름과 마지막이름들이 바꾸어 진 파일의 내용을 인쇄하시오.

    ㄹ. 파일의 끝에 THE END를 붙이시오.

제 5 장. awk편의프로그람: UNIX도구로서의 awk

제 1 절. awk란 무엇인가

awk는 자료다루기와 보고서(report)작성에 사용되는 프로그람작성언어이다. 자료는 표준입력, 한개이상의 파일 또는 한개의 프로쎄스로부터 들어 온다. awk는 간단한 조작을 위해 지령행에서 사용될수도 있고 혹은 큰 응용프로그람으로도 쓸수 있다. awk는 자료를 처리할수 있으므로 쉘스크립트와 작은 자료기지관리에 사용되는 필수적인 도구이다.

awk는 규정된 패턴들을 일치시키는 행을 탐색하고 그 행들에서 선택된 동작(괄호안에 놓이는)들을 수행하면서 첫 행에서 마지막끝까지 한행씩 파일(혹은 입력)을 조사한다. 만일 규정된 동작을 가지지 않는 패턴이 있으면 그 패턴이 일치되는 모든 행들을 표시한다. 패턴이 없이 동작하면 그에 의해 규정된 모든 입력행들이 실행된다.

1. awk이름의 유래

awk는 이 언어개발자들인 Alfred Aho, Brain kernighan, Peter Weinberger의 마지막이름의 첫 글자들에서 딴것이다. 그것은 wak나 kaw 등으로도 부를수 있었지만 종당에는 awk로 하였다.

2. awk에는 어떤것들이 있는가

awk의 판본들에는 낡은 awk, 새로운 awk, gnu(gawk), POSIX awk 등 여러가지가 있다. awk는 처음에 1977년, 다음에 1985년에 나왔으며 초기제품들은 awk가 보다 큰 프로그람들을 조작할수 있도록 개선되였다. 추가적인 기능들은 사용자정의함수, 동적정규식, 다중입력파일의 처리 등이다. 대부분의 체계들에서 지령은 낡은 판본에서 awk, 새 판본에서 nawk, gnu판본[1]에서 gawk이다.

제 2 절. awk의 양식

awk프로그람은 awk지령, 인용부호(혹은 파일)안에 놓이는 프로그람지령, 입력파일이름으로 구성된다. 만일 입력파일이 지정되지 않으면 입력은 표준입력(stdin)인 건반으로부터 들어 온다.

awk지령들은 패턴, 동작 혹은 그것들의 조합으로 구성된다. 패턴은 몇개의 형식표현으로 구성되는 명령문이다. 예약어 if가 없어도 표현을 평가할 때 단어 if가 요구되면 그것은 패턴으로 된다. 동작들은 반두점이나 행바꾸기로 구별되며 괄호안에 놓이는 하나이상의 명령문으로 이루어 진다. 패턴들은 대괄호안에 놓일수 없으며 빗선문자 /안에 놓이는 정규식이나 awk에 의해 제공되는 한개이상의 많은 연산자들로 구성된다. awk

---

1. Sco UNIX에서 새 판본은 awk로, Linux에서는 gnu판본을 awk로 한다. 이 본문은 새로운 awk 즉 nawk와 관련된다. Gnu제품 gawk는 nawk와 완전히 웃방향호환성을 가진다.

지령은 지령행 혹은 awk스크립트들에서 입력될수 있다. 입력행들은 파일, 파이프, 표준 입력으로부터 입력될수 있다.

1. 파일로부터의 입력

다음의 실례에서 퍼센트문자(%)는 C쉘재촉문이다.

형식

```
% nawk 'pattern' 파일이름
% nawk '{action}' 파일이름
% nawk 'pattern' {action}' 파일이름
```

employee라고 하는 표본파일이 있다.

실례 5-1

```
% cat employees
Tom Jones      4424   5/12/66   543354
Mary Adams     5346   11/4/63   28765
Sally Chang    1654   7/22/54   650000
Billy Black    1683   9/23/44   336500

% nawk  '/Mary/' employees
Mary Adams     5436   11/4/63   28765
```

설명

nawk는 패턴 Mary를 포함하는 모든 행들을 인쇄한다.

실례 5-2

```
% cat employees
Tom Jones      4424   5/12/66   543354
Mary Adams     5346   11/4/63   28765
Sally Chang    1654   7/22/54   650000
Billy Black    1683   9/23/44   336500

% nawk '{print $1}' employees
Tom
Mary
Sally
Billy
```

설명

nawk는 마당이 행의 왼쪽 끝에서 시작하고 공백으로 구분되는 파일 emplo~yees의 첫 마당을 인쇄한다.

실례 5-3

% cat employees

Tom Jones	*4424*	*5/12/66*	*543354*
Mary Adams	*5346*	*11/4/63*	*28765*
Sally Chang	*1654*	*7/22/54*	*650000*
Billy Black	*1683*	*9/23/44*	*336500*

% **nawk '/Sally/{print $1, $2}' employees**
Sally Chang

설명

행이 패턴 sally를 포함할 때에만 nawk는 파일 employees의 첫번째와 두번째 마당을 인쇄한다. 마당분리기호는 공백이다.

2. 지령으로부터의 입력

UNIX지령이나 지령들로부터의 출력은 처리를 위해 awk에 련결될수 있다. 일반적으로 쉘프로그람들은 지령을 조종하기 위해 awk를 사용한다.

형식

% 지령 | nawk ‘pattern’

% 지령 | nawk ‘{action}

% 지령 | nawk ‘pattern{action}’

실례 5-4

1. **0% df | nawk •$4 > 75000'**

   *_/oracle r/dey/dsA/cOtOd057):.-350780 blocks 105756 files_*
   *_/opt (/dev/cisk/c0t0cl058):1943994 blocks 49187 files_*

2. % rusers [nawk '/root$/{print $1}’
   owl
   crow
   bluebird

설명

1. df지령은 파일체계에서 디스크의 빈 공간을 알려 준다. df지령의 출력은 nawk(새로운 awk)에로 파이프된다. 4번째 마당이 75000개의 블로크보다 더 크면 행이 인쇄된다.
2. rusers지령은 망상의 원격콤퓨터에 가입된 사용자들을 인쇄한다. rusers지령의 출력은 nawk에 입력으로 파이프된다. 첫 마당은 정규식 root가 행의 끝($)에서 일치되면 인쇄된다. 즉 모든 장치이름들은 root가 가입되는 곳에서 인쇄된다.

제 3 절. 출력형식화

1. print함수

awk지령의 동작부분은 대괄호안에 놓인다. 지정된 동작이 없고 패턴이 일치되면 awk는 일치된 행들을 현시장치에 인쇄하는 기정동작을 취한다. print함수는 특별한 형식화를 요구하지 않는 간단한 출력을 인쇄하기 위해 사용한다. 보다 다양한 형식화에 대하여서는 printf와 sprintf함수를 사용한다. 사용자가 C언어를 잘 안다면 printf와 sprintf가 어떻게 동작하는가를 이미 알고 있을것이다.

print함수도 awk의 동작부분에서 {print}로써 사용할수 있다. print함수는 인수로 변수, 계산된 값, 혹은 문자렬내용들을 받는다. 문자렬은 2중괄호안에 있어야 한다. 반점은 인수들을 구별하는데 사용한다. 반점이 없으면 인수들은 서로 련결된다. 반점은 출력마당분리기호(OFS)의 값으로 되며 기정값으로는 공백이다.

print함수의 출력은 다른 프로그람에로 방향바꾸기되거나 파이프될수 있으며 그 프로그람의 출력은 awk에 인쇄를 위해 파이프될수 있다(24페지의 《파이프》와 22페지의 《방향바꾸기》를 참고하기 바란다.).

실례 5-5

```
% date
Wed Jul   28     22:23:16     PDT      2001
% date | nawk '{ print "Month: " $2 "\ nYear: " , $6 }'
Month: Jul
Year: 2001
```

설명

UNIX의 data지령출력은 nawk에 파이프된다. 문자렬 Month:가 인쇄되고 그 뒤에 두번째 마당, 행바꾸기문자 \ n, Year:를 포함하고 있는 문자렬 그리고 여섯번째 마당($6)이 놓인다.

탈출문자(escape sequence) 탈출문자는 거꿀빗선(\ )과 문자 혹은 수자로 표현된다. 이것들은 타브(tab), 행바꾸기, 행의 첫 머리에로 이동 등을 표현하기 위해 문자렬에서 사용된다.

표 5-1. Print탈퇴지령

탈퇴지령	의미
\ b	뒤방향지우기
\ f	행의 첫 머리로 이행
\ n	행바꾸기
\ r	다음행으로 이행]
\ t	타 브

(표계속)

탈퇴지령	의미
\ o47	8진수값 47 즉 단일인용부호
\ c	C는 임의의 다른 문자 즉 \ ”를 표시한다.

실례 5-6

Tom Jones	*4424*	*5/12/66*	*543354*
Mary Adams	*5346*	*11/4/63*	*28765*
Sally Chang	*1654*	*7/22/54*	*650000*
Billy Black	*1683*	*9/23/44*	*336500*

% nawk -/Sally/{print “\ t\ tHave a nice day, “ $1, $2 “\ !”}. employees
Have a nice day, Sally Chang!

설명

행이 패턴 saalyrl를 포함하고 있으면 print함수는 2개의 타브, 문자렬 Have a nice day, 첫번째 마당($1이 sally인)과 두번째 마당($2이 change인), 그뒤에 2개의 감탄부호 !!를 포함하고 있는 문자렬을 인쇄한다.

2. OFMT 변수

흔히 수자들을 인쇄할 때 그 형식을 조종할 필요가 제기된다. 일반적으로 이 기능을 printf함수로 하지만 특수한 awk변수인 OFMT를 설정하여 print함수를 리용할 때 수자들의 인쇄를 조종할수 있다. 기정값은 《%.6g》이다. 즉 소수점 오른쪽으로 6개의 수자들이 인쇄된다(다음부분에서는 이 값을 어떻게 변화시킬수 있는가를 설명한다.).

실례 5-7

% nawk 'BEGIlI{OFMT=“%.2f”; print 1.2456789, 12E-2} '
1.25 *0.12*

설명

OFMT변수는 류동소수(f)들이 소수점뒤에 2개의 수자가 붙어 인쇄되도록 한다. 퍼센트기호(%)는 형식이 규정되여 있다는것을 지적한다.

3. printf 함수

출력을 인쇄할 때 렬이 질서 있게 놓이도록 마당들사이에 공백의 수들을 규정할 필요가 있다. 타브를 가진 print함수가 항상 요구되는 출력을 담보할수는 없으므로 print 함수는 필요한 출력을 형식화하기 위해 사용된다.

printf함수는 형식화된 문자렬을 C언어에서의 printf명령문과 같이 표준출력으로

되돌려 준다. printf명령문은 형식지정과 변경자들로 이루어 진 인용부호안에 있는 조종문자렬로 구성된다. 조종문자렬뒤에 반점으로 분리되는 표현들이 놓이는데 이것들은 조종문자렬에서 표시된 내용에 따라 형식화된다. print함수와는 달리 printf함수는 행바꾸기를 제공하지 않는다. 탈출문자 \ n은 행바꾸기가 요구될 때 주어 져야 한다. 매 퍼센트문자와 형식지적자에 대해서는 해당한 인수가 있어야 한다. 문자로서 퍼센트기호를 인쇄하자면 2개의 퍼센트기호 즉 %%가 사용되여야 한다. printf변환문자목록에 대하여서는 표 5-2, printf변경자에 대하여서는 표 5-3에서 보여 주었다. 형식지적자들은 앞에 퍼센트문자를 붙인다. printf형식지적자들의 목록을 표 5-4에서 주었다. 어떤 인수들을 인쇄할 때 출력이 인쇄되는 위치를 마당(field)이라고 하며 마당의 너비(width)는 그 마당에 포함된 문자수이다.

다음의 실례에 있는 파이프문자(수직막대기)가 printf문자렬의 일부분이면 본문의 부분이며 형식이 시작되는것과 끝나는것을 지적하기 위해 사용된다.

실례 5-8

1. **% echo “UNIX” | nawk ‘ {printf “|%-15s|\ n”, $1}’**

   (Output)

   |*UNIX* |

2. **% echo “UNIX” | n»wk ‘{ printf “|%15s|\ n”, $1}’**

   (Output)

   | UNIX|

설명

1. echo지령의 출력 UNIX는 nawk에 파이프된다. printf함수는 조종문자렬을 포함하고 있다. 퍼센트문자는 printf가 15개의 공백, 수직막대기안에 놓여있는 행바꾸기로 끝나는 왼쪽정돈된 문자렬을 인쇄하도록 한다. %기호뒤에 횡선 –는 왼쪽정돈을 가리킨다. 조종문자렬뒤에 반점과 $1이 놓인다. 문자렬 UNIX는 조종문자렬에 있는 형식지정에 따라 형식화된다.
2. 문자렬 UNIX는 오른쪽정돈되고 수직막대기안에 놓이는 15개의 공백문자렬과 그뒤에 오는 행바꾸기문자로 인쇄된다.

실례 5-9

% cat employees

Tom Jones	*4424*	*5/12/66*	*543354*
Mary Adams	*5346*	*11/4/63*	*28765*
Sally Chang	*1654*	*7/22/54*	*650000*
Billy Black	*1683*	*9/23/44*	*336500*

% **nawk ‘ {printf “The name is: % -15s ID %8d\ n”, $1, $3} ’ employees**

The name is Tom *ID is4424*
The name is Mary *ID is5346*
The name is Sally *ID is1654*
The name is Billy *ID is1683*

설명

인쇄하려는 문자렬은 인용부호안에 있다. 첫 형식지적자는 %-15s이다. 조종문자렬의 닫는 괄호뒤에 반점이 있고 그 오른쪽에 인수 $1이 있다. 퍼센트문자는 형식지적자를 나타낸다. 즉 횡선(-)은 왼쪽정돈을 의미하며 15s는 15개의 공백문자렬을 의미한다. 이 위치에서 왼쪽정돈된 15개의 공백문자렬과 문자렬 ID is, 수자가 인쇄된다.

%8d형식은 $2의 10진값(옹근수)이 문자렬안의 위치에서 인쇄된다는것을 지적한다. 수자는 오른쪽으로 정돈되며 8개의 공백을 가진다. 인용부호안에 있는 문자렬과 표현식을 선택적으로 괄호안에 넣을수 있다.

표 5-2. printf변환문자

변환문자	정의
c	문자
s	문자렬
d	10진수
ld	긴 10진수
u	부호 없는 10진수
lu	긴 부호 없는 10진수
x	16진수
lx	긴 16진수
o	8진수
lo	긴 8진수
e	류동소수점수(e표기)
f	류동소수점수
g	e나 f변환을 리용하는 류동소수점수(공백을 가진다.)

표 5-3. printf변경자

문자	정의
–	왼쪽 정렬변경자
#	8진수형식의 옹근수앞에 0으로 표시한다. 16진수형식의 옹근수들앞에 ox를 붙인다.
+	d, e, f, g를 사용하는 변환에서 옹근수를 + 혹은 –기호와 함께 표시한다.
0	표시되는 값은 공백대신 령으로 된다.

표 5-4. print형식지적자

형식지적자	무엇을 수행하는가
	given x='A', y=15, z=2.3, and $1=Bob Smith
%c	하나의 ASCII문자를 인쇄한다. printf("The character is %c\ n",x)는 The character is A를 인쇄한다.
%d	10진수를 인쇄한다. printf("The boy is %d years old \ n", y)는 The boy is 15 years old를 인쇄한다.
%e	수자를 e표기로 인쇄한다. printf("x is %e\ n, z)는 z is 2.3e+01을 인쇄한다.
%f	류동소수점수를 인쇄한다. printf("z is %f\ n",2.3*2)는 z is 4600000을 인쇄한다.
%o	8진수값을 인쇄한다. printf("y is %o\ n,y)는 z is 17을 인쇄한다.
%s	문자렬을 인쇄한다. printf("The name of the culprit is %s\ n",$1)은 The name of the culprit is Bob Smith를 인쇄한다.
%x	16진수값을 인쇄한다. printf("y is %x\ n,y)는 x is f를 인쇄한다.

제 4 절. 파일내부로부터 실행되는 awk지령

awk지령들이 파일에 배치되면 -f추가선택은 awk파일의 이름과 그뒤에 놓이는 처리하려는 입력파일의 이름과 함께 사용된다. 한개의 레코드가 awk의 완충기에 넣어 지고 awk파일에 있는 매 지령들이 검사되여 그 레코드에 대하여 실행된다. awk는 첫 레코드에 대하여 완료한후 그 레코드를 지우고 다음의 레코드를 완충기에 다시 써넣는 식으로 계속한다. 만일 동작이 패턴에 의해 조종되지 않으면 기정적인 동작은 전체 레코드를 인쇄하는것이다. 패턴에 그와 관련된 동작이 없으면 기정으로 패턴은 입력행을 일치시키는 레코드를 인쇄한다.

실례 5-10

```
(The Database)
$1        $2        $3      $4        $5
Tom       Jones     4424    5/12/66   543354
Mary      Adams     5346    11/4/63   28765
Sally     Chang     1654    7/22/54   650000
Billy     Black     1683    9/23/44   336500

% cat awkfile
1   /"Mary/{print "Hello Maryl"I
2   {print $1, $2, $3}

    % nawk -f awkfile employees
```

Tom Jones 4424
Hello Mary!
Mary Adams 5346
Sally Chang 1654
Billy Black 1683

설명

1. 레코드가 정규식 Mary로 시작되면 문자렬 Hello Mary!가 인쇄된다. 동작은 그앞에 있는 패턴에 의해 조종된다. 마당들은 공백에 의해 구분된다.
2. 매 레코드의 첫번째, 두번째, 세번째 마당을 인쇄한다. 동작을 조종하는 패턴이 없으므로 동작은 매행에 대하여 진행된다.

제 5 절. 레코드와 마당

1. 레코드

awk는 입력자료를 끝 없는 문자렬로가 아니라 하나의 형식과 구조를 가진 문자렬로 취급한다. 기정적으로는 매행을 레코드라고 하며 행바꾸기로써 완료된다.

레코드분리기호 기정적으로 입력과 출력레코드분리기호(행분리기호)는 복귀(CR)로써 각각 awk의 내부변수, ORS, RS에 기억된다. ORS와 RS값은 오직 제한된 방식에서만 변화될수 있다.

$0변수 전체 레코드는 awk에 의해 $0으로 참조된다($0이 교환 혹은 값주기에 의해 변화될 때 마당의 수인 NF값이 변화될수 있다.). 행바꾸기문자의 값은 awk의 내부변수 RS에 기억되는데 기정적으로는 복귀(CR)이다.

실례 5-11

```
% cat employees
Tom Jones       4424    5/12/66    543354
Mary Adams      5346    11/4/63    28765
Sally Chang     1654    7/22/54    650000
Billy Black     1683    9/23/44    336500

% nawk '{print $0}'   employees
Tom Jones       4424    5/12/66    543354
Mary Adams      5346    11/4/63    28765
Sally Chang     1654    7/22/54    650000
Billy Black     1683    9/23/44    336500
```

설명

nawk변수 $0은 현재레코드를 기억한다. 그것을 현시장치에 출력한다. 기정적으로 nawk는 지령이 다음과 같으면 레코드를 인쇄한다.

```
%nawk '{print}' employees.
```

NR변수 매 레코드번호는 awk내부변수 NR에 기억된다. NR값은 레코드가 처리된후에 하나씩 증가된다.

실례 5-12

```
% cat employees
Tom Jones        4424    5/12/66    543354
Mary Adams       5346    11/4/63    28765
Sally Chang      1654    7/22/54    650000
Billy Black      1683    9/23/44    336500

% nawk ' {print $0} '     employees
1  Tom Jones     4424    5/12/66    543354
2  Mary Adams    5346    11/4/63    28765
3  Sally Chang   1654    7/22/54    650000
4  Billy Black   1683    9/23/44    336500
```

설명

매 레코드 $0은 파일에 기억된것과 같이 인쇄되며 레코드번호 NR가 앞에 붙는다.

2. 마당

매 레코드는 fields라고 하는 단어들로 구성되며 기정적으로는 공백 즉 빈 공백이나 타브(tab)로 구분한다. 이 매 단어를 마당이라고 하며 awk는 내부변수 NF에 마당번호를 기억한다. NF의 값은 행에 따라 다르지만 그 한계는 실행에 의존하는데 표준적으로는 한행당 100개의 마당이다. 새로 마당을 작성할수 있다. 다음의 실례는 4개의 레코드(행)들과 5개의 마당(렬)들을 가진다. 매 레코드는 $1로써 표현되는 첫 마당에서 시작되며 그다음에 두번째 마당 $2에로 이동하는 방법으로 계속한다.

실례 5-13

```
(Fields are represented by a dollar sign and the number of the field )
(The Database)
$1       $2       $3      $4        $5
Tom      Jones    4424    5/12/66   543354
Mary     Adams    5346    11/4/63   28765
Sally    Chang    1654    7/22/54   650000
Billy    Black    1683    9/23/44   336500

% nawk ' {print HR, $1, $2, $5} '   employees

1  Tom Jones      543354
2  Mary Adams     28765
```

```
3   Sally Chang      650000
4   Billy Black      336500
```

설명

nawk는 파일의 매행의 첫번째, 두번째, 다섯째 마당(렬)과 레코드번호(NR)를 인쇄한다.

실례 5-14

```
% nawk  '{print $0, MF}'  employees
Tom Jones        4424    5/12/66    543354    5
Mary Adams       5346    11/4/63    28765     5
Sally Chang      1654    7/22/54    650000    5
Billy Black      1683    9/23/44    336500    5
```

설명

nawk는 파일에서 매 레코드($0)와 그뒤에 마당번호를 인쇄한다.

3. 마당분리기호

입력마당분리기호 awk의 내부변수 FS는 입력마당분리기호값을 가진다. fS의 기정값을 사용할 때 awk는 마당들을 공백이나 타브로 분리시켜 처음에 있는 빈 칸이나 타브들을 지우기한다. FS는 BEGIN명령 또는 지령행에서 새값을 그에 대입하는 방법으로 변화시킬수 있다. 여기서는 지령행에 새값을 대입한다. 지령행에서 FS의 값을 변화시키자면 -F추가선택과 그뒤에 새로운 식별자를 나타내는 문자를 사용한다.

지령행에서 마당분리기호변경

실례 5-15

```
% cat employees
Tom Jones:4424: 5/12/66: 543354
Mary Adams:5346:11/4/63:28765
Sally Chang:1654:7/22/54:650000
Billy Black:1683:9/23/44:336500

% nawk -F: '/Tom Jones/{print $1, $2}'  employees2
 Tom Jones   4424
```

설명

-F추가선택은 지령행에 입력마당분리기호값을 다시 대입하기 위해 사용한다. -F추가선택뒤에 직접 두점(:)이 놓이면 nawk은 employees파일에서 마당들을 분리시키는 두점들을 탐색한다.

여러개의 마당분리기호사용 사용자는 하나이상의 입력분리기호를 지정할수 있다. 하나이상의 문자를 마당분리기호 FS를 위해 사용하면 문자렬은 정규식이며 중괄호([]) 안에 놓인다. 다음의 실례에서는 마당분리기호가 공백, 두점 혹은 타브이다(awk의 이전판본은 이 특성을 제공하지 않는다.).

실례 5-16

% **nawk -F-[i\ tp -{print $1, $2, $3}- employees**
Tom Jones 4424
Mary Adams 5346
Sally Chang 1654
Billy Black 1683

설명

-F추가선택뒤에는 중괄호안에 있는 정규식이 놓인다. 공백이나 두점, 타브가 나타나면 nawk는 마당분리기호로 그 문자를 사용한다. 표현은 인용부호안에 넣어 쉘이 메타문자들과 그자체를 혼돈하지 않게 한다(쉘이 파일이름확장을 위해 중괄호를 사용한다는것을 명심해야 한다.).

출력마당분리기호 기정적인 출력마당분리기호는 하나의 공백이며 awk의 내부변수 OFS에 기억된다. 앞의 모든 실례들에서는 print명령문을 사용하여 현시장치에 출력하였다. print명령문에서 마당을 분리하는데 리용되는 반점은 OFS가 설정된것들로 된다. 기정값이 사용되면 $1과 $2사이에 삽입된 반점은 하나의 공백으로 평가되며 print함수는 그것들사이에 공백으로 넣어 마당들을 인쇄한다. OFS는 변경시킬수 있다.

마당을 분리하는데 반점을 사용하지 않으면 혼돈된다. OFS는 반점이 마당을 구분하지 않으면 해석되지 않는다.

실례 5-17

% **cat employees2**
Tom Jones:4424 -.5/12/66:543354
Mary Adams:5346:11/4/63:28765
Sally Chang:1654:7/22/54:650000
Billy Black:1683:9/23/44:336500

(The Command Line)

% **nawk -F; ' /Tom Jones/{print $1, $2, $3, $4} ' employees2**
Tom Jones 4424 5/12/66 543354

설명

출력마당분리기호, 공백은 naw의 OFS변수에 기억된다. 마당사이의 반점은 OFS에 기억된것으로 된다. 마당들은 공백으로 구분된 표준출력으로 출력된다.

실례 5-18

% **nawk -F: ' /Tom Jones/(print $1 $2 $3 $4) ' employees2**
Tom Jones44245/12/66543354

설명

$0변수는 현재의 레코드를 입력파일에 있는것과 꼭같이 기억한다.
레코드는 as-is로 인쇄된다.

제 6 절. 패턴과 동작

1. 패턴

awk패턴들은 awk가 입력의 행에 대해 어떤 동작을 할수 있는가를 조종한다. 패턴은 정규식, 참과 거짓조건들을 나타내는 표현 혹은 그의 조합으로 이루어 진다. 기정동작은 표현이 참의 조건을 나타내는 모든 행을 인쇄하는것이다. 패턴표현을 읽을 때 if명령문이 리용된다. if가 리용될 때 그 주위에 대괄호({})가 없어도 된다. if가 리용되면 그것은 동작명령문으로 되며 문법도 달라 진다(154페지에서 《조건명령문》을 참고).

실례 5-19

% cat employees

Tom Jones	*4424*	*5/12/66*	*543354*
Mary Adams	*5346*	*11/4/63*	*28765*
Sally Chang	*1654*	*7/22/54*	*650000*
Billy Black	*1683*	*9/23/44*	*336500*

(The Command Line)

1 **nawk ' /Tom/ ' employees**

Tom Jones	*4424*	*5/12/66*	*543354*

2 **nawk '$3 < 4000- employees**

Sally Chang	*1654*	*7/22/54*	*650000*
Billy Black	*1683*	*9/23/44*	*336500*

설명

1. 패턴 Tom이 입력파일에서 일치되면 그 레코드가 인쇄된다. 기정동작은 명백히 서술되지 않을 때 그 행을 인쇄하는것이다. 그것은 다음과 같다.

   Nawk '$0-/ Tom/{print$0}' employees
2. 세번째 마당이 4000이하이면 레코드는 인쇄된다.

2. 동작

동작은 대괄호안에 놓이고 반두점으로 구분되는 명령문이다.[2] 만일 패턴이 동작앞에 놓이면 언제 동작이 수행되여야 하는가를 알려 준다. 동작은 간단한 명령문이나 복잡한 명령문모임일수 있다. 명령문들은 그 행우에 놓일 때 반두점이나 행바꾸기문자에 의해 분리된다.

---

2. awk의 어떤 판본에서 동작은 반두점 혹은 행바꾸기로 구분되여야 하며 대괄호안에 있는 명령문들도 반두점이나 행바꾸기로 구분되여야 한다. sVR4의 nawk는 동작안에서 명령문을 식별하기 위해 반두점이나 행바꾸기를 사용하지만 동작을 구분하는데 반두점을 리용하지 않는다. 실례로 련속적인 두 동작은 반두점을 요구하지 않는다. 즉

   nawk '/Tom/{print "hi Tom"};{x=5}' file

형식

{action}

실례 5-20

```
{ print $1, $2 }
```

설명

동작은 마당 1과 2를 인쇄하는것이다. 패턴들은 동작과 련관될수 있다. 동작들이 대괄호안의 명령문들이라는것을 명심하여야 한다. 패턴은 첫번째 열린 대괄호로부터 첫번째 닫는 대괄호안의 동작을 조종한다. 동작이 패턴뒤에 놓이면 첫번째 여는 대괄호는 패턴과 같은 행에 있어야 한다. 패턴들은 대괄호안에 놓이지 않는다.

형식

```
pattern{ action statement; action statement; etc. }
or
  pattern{
action statement
action statement
}
```

실례 5-21

```
% nawk '/Tom/{print "Hello there, "   $1}' employees
Hello there, Tom
```

설명

레코드가 패턴 Tom을 가지고 있으면 문자렬 Hello there, Tom을 인쇄한다. 동작이 없는 패턴은 그 패턴과 일치되는 모든 행들을 다 표시한다. 문자렬일치패턴은 빗선(/)으로 둘러 싸인 정규식을 포함하고 있다.

제 7 절. 정규식

awk에서의 정규식은 빗선안의 문자렬들로 이루어 진 패턴이다. awk는 정규식들을 일련의 방법으로 변화시키기 위해 정규표현메타문자(egrep와 같은)를 사용한다. 입력행에 있는 문자렬이 정규식과 일치되면 결과조건은 참이며 표현과 관련된 그 어떤 동작이 실행된다. 동작이 규정되지 않고 입력행이 정규식과 일치되면 레코드를 인쇄한다(표 5-5).

실례 5-22

```
% nawk  '/Mary/'  employees
```

Mary Adams 5345 11/4/63 28765

설명

정규식패턴 Mary를 가지고 있는 employees파일에서 모든 행들을 표시한다.

실례 5-23

% nawk •/Mary/{print $1, $2}' employees
Mary Adams

설명

employees파일에서 정규식패턴 Mary를 포함하는 모든 행의 첫번째와 두번째 마당들이 인쇄된다.

표 5-5. awk정규표현메타문자

메타문자	의 미
^	문자렬의 시작에서 일치시킨다.
$	문자렬의 끝에서 일치시킨다.
.	한개의 문자를 일치시킨다.
*	임의의 수의 앞의 문자를 일치시킨다.
+	한개이상의 앞의 문자를 일치시킨다.
?	령 혹은 하나앞의 문자를 일치시킨다.
[ABC]	문자들의 모임 즉 A, B, C에서 임의의 하나의 문자를 일치시킨다.
[^ABC]	문자들의 모임 즉 A, B, C에 포함되지 않는 문자들을 일치시킨다.
[A-Z]	[A-Z]의 범위에서 임의의 문자를 일치시킨다.
A|B	A나 B중의 하나를 일치시킨다.
(AB)+	AB에 대한 하나이상의 모임을 일치시킨다.
\ *	문자별표를 일치시킨다.
&	탐색문자렬에 있는것을 표시하기 위해 교체문자렬에서 사용된다.

실례 5-24

% nawk '/*Mary/' employees

Mary Adams 5346 11/4/63 28765

설명

정규식 Mary로 시작하는 employees파일의 모든 행들이 표시된다.

실례 5-25

% nawk '^[A-Z] [a-z]+ /' employees

Tom Jones	*4424*	*5/12/66*	*543354*
Mary Adams	*5346*	*11/4/63*	*28765*
Sally Chang	*1654*	*7/22/54*	*650000*
Billy Black	*1683*	*9/23/44*	*336500*

설명

employees파일에서 대문자로 시작하며 그뒤에 하나 또는 그이상의 작은 글자와 공백이 놓이는 모든 행들을 표시한다.

1. 대조연산자(match operator)

대조연산자 ~는 레코드나 마당안에 있는 표현을 일치시키는데 사용된다.

실례 5-26

% cat employees

Tom Jones	*4424*	*5/12/66*	*543354*
Mary Adams	*5346*	*11/4/63*	*28765*
Sally Chang	*1654*	*7/22/54*	*650000*
Billy Black	*1683*	*9/23/44*	*336500*

% nawk '$1 - /[Bb]ill/' employees

Billy Black	*1683*	*9/23/44*	*336500*

설명

첫 마당에서 Bill이나 bill과 일치하는 모든 행들이 표시된다.

실례 5-27

% nawk "$l !- /ly$/' employees

Tom Jones	*4424*	*5/12/66*	*543354*
Mary Adams	*5346*	*11/4/63*	*28765*

설명

첫 마당의 끝에 여러개의 Ly가 있을 때 그것과 일치하지 않는 모든 행을 표시한다.

제 8 절. 스크립트파일안에서의 awk지령

다중awk패턴 혹은 동작명령문이 있을 때 스크립트에 명령들을 써넣는것이 훨씬 쉽다. 스크립트는 awk지령과 명령문을 가진 파일이다. 명령문들과 동작들이 같은 행에 있을 때는 반두점으로 구분된다. 명령문이 개별적인 행들에 있으면 반두점이 필요 없다. 동작앞에 패턴이 있으면 열린 대괄호가 패턴과 같은 행에 있어야 한다. 설명문앞에는 # 기호가 놓인다.

실례 5-28

% **cat employees**
Tom Jones:4424:5/12/66:54335
Mary Adams:5346:11/4/63:28765
Billy Black:1683.-9/23/44:336500
Sally Chang:1654:7/22/54:65000
Jose Tomas:1683:9/33/44.33650

(The Awk Script)
% **cat info**
1. *# My first awk script by Jack Sprat*
   # Script name: info; Date: February 28, 2001
2. */Tom/{print "Tom's birthday is "$3}*
3. */Mary/{print NR, $0}*
4. */^Sally/{print "Hi Sally. " $1 " has a salary of $" $4 "."}*
   # End of info script

(The Command Line)
5. % **nawk -F: -f info employees2**
   Tom's birthday is 5/12/66
2 Mary Adams:5346:11/4/63:28765
   Hi Sally. Sally Chang has a salary of $65000

설명

1. 이것은 설명행이다.
2. 정규식 Tom이 입력행과 일치되면 문자렬 Tom's birthday와 세번째 마당($3)의 값이 인쇄된다.
3. 정규식 Mary가 입력행과 일치되면 동작블로크는 NR와 현재레코드번호 그리고 레코드를 인쇄한다.
4. 정규식 Sally를 입력행의 시작에서 찾으면 문자렬 Hi Sally가 인쇄되는데 그 뒤에는 첫 마당($1)의 값과 $문자렬 has a salary of $와 네번째 마당($4)의 값이 인쇄된다.
5. nawk지령뒤에 –F:추가선택을 놓아 두점이 마당분리기호라는것을 지적한다. 그리고 -f추가선택은 awk스크립트의 이름의 앞에 놓인다. awk는 info 파일로부터 지령을 읽는다. 입력파일 employees2가 다음에 놓인다.

제 9 절. 개괄

이 절의 실례들은 datafile이라는 표본자료기지를 사용하였다. 자료기지에서 입력마당분리기호 FS는 기정적으로는 공백이다. 마당의 수 NF는 8이다. 이 수는 행에 따라 변할수 있지만 이 파일에서는 그 수가 고정된다. 레코드분리기호 RS는 파일의 매행을 분리시키는 행바꾸기이다. awk는 NR변수에 매 레코드번호를 기억시킨다. 출력마당분리기호 OFS는 공백이다. 반점이 마당구별에 사용되면 행이 인쇄될 때 인쇄된 매 마당이 공백에 의해 구별된다.

1. 간단한 패턴대조

```
% cat datafile
northwest      NW    Joel Craig       3.0   .98   3    4
western        WE    Sharon Kelly     5.3   .97   5   23
southwest      SW    May Chin         2.7   .8    2   18
southern       SO    Suan Chin        5.1   .95   4   15
southeast      SE    Derek Johnson    4.0   .7    4   17
eastern        EA    Susan Beal       4.4   .84   5   20
northeast      NE    TJ Nichols       5.1   .94   3   13
north          NO    Val hultz        4.5   .89   5    9
central        CY    Sheri Watson     5.7   .94   5   13
```

실례 5-29

```
nawk '/west/'      datafile
northwest     NW    Joel Craig     3.0   98    3   4
western       WE    Sharon Kelly   5.3   .97   5   23
southwest     SW    Chris Foster   2.7   .8    2   18
```

설명

패턴 west를 가지는 모든 행들이 인쇄된다.

실례 5-30

```
nawk '/^orth/' datafile
northwest   NW   Joel Craig      3.0   98    3   4
northeast   NE   TJ Nichols      5.1   .94   3   13
north       NO   Val Shultz      4.5   .89   5   9
```

설명

패턴 north로 시작하는 모든 행들이 인쇄된다.

실례 5-31

```
nawk ' /'• {m3\   so} / ' datafile
northwest   NW   Joel Craig      3.0   98    3   4
southwest   SW   Chris Foster    2.7   .8    2   18
southern    SO   May Chin        5.1   .95   4   15
southeast   SE   Derek Johnson   4.0   .7    4   17
northeast   NE   TJ Nichols      5.1   .94   3   13
north       NO   Val Shultz      4.5   .89   5   9
```

설명

패턴 no 혹은 so로 시작되는 모든 행들이 인쇄된다.

2. 간단한 동작

실례 5-32

nawk '{printf $3,$2}' datafile
Joel NW
Sharon WE
Chris SW
May SO
Dreck SE
Susan EA
TJ NE
Val NO
Sheri CT

설명

출력마당분리기호 OFS는 기정값이 공백이다. $3과 $2사이의 반전은 OFS의 값으로 변환된다. 세번째 마당과 그뒤에 한개의 공백, 두번째 마당이 인쇄된다.

% cat datafile

northwest	NW	Joel Craig	3.0	.98	3	4
western	WE	Sharon Kelly	5.3	.97	5	23
southwest	SW	May Chin	2.7	.8	2	18
southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Derek Johnson	4.0	.7	4	17
eastern	EA	Susan Beal	4.4	.84	5	20
northeast	NE	TJ Nichols	5.1	.94	3	13
north	NO	Val hultz	4.5	.89	5	9
central	CY	Sheri Watson	5.7	.94	5	13

실례 5-33

nawk '{print $3 $2}' datafile
JoelNW

설명

세번째 마당은 두번째 마당앞에 놓인다. 반점이 마당 $3과 $2를 구별하지 못하므로 출력은 마당들사이에 공백없이 표시된다.

실례 5-34

nawk ' print $1' datafile
nawk:syntax error at source line 1
context is

```
>>> print <<< $1
nawk: bailing out at source line 1
```

설명

이것은 nawk(새로운 awk)오유통보문이다. nawk오유통보문들은 이전 판본 awk 통보문보다 훨씬 더 구체적이다. 이 프로그람에서 동작명령문은 대괄호를 요구한다.

실례 5-35

```
awk ' print $1' datafile
awk: syntax error near line 1
awk: bailing out near line 1
```

설명

이것은 이전 awk의 오유통보문이다. 이전 awk프로그람들은 거의 대부분이 우와 같은 통보문을 내보내기때문에 오유수정이 어렵다. 동작명령문에 대괄호가 요구된다.

실례 5-36

```
nawk   '{print $0}' datafile
northwest     NW     Joel Craig       3.0    .98    3    4
western       WE     Sharon Kelly     5.3    .97    5    23
southwest     SW     Chris Foster     2.7    .8     2    18
southern      SO     May Chin         5.1    .95    4    15
southeast     SE     Derek Johnson    4.0    .7     4    17
eastern       EA     Susan Beal       4.4    .84    5    20
northeast     NE     TJ Nichols       5.1    .94    3    13
north         NO     Val Shultz       4.5    .89    5    9
central       CT     Sheri Watson     5.7    .94    5    13
```

설명

매 레코드가 인쇄된다. $0은 현재의 레코드를 가진다.

실례 5-37

```
nawk '{print "Number of fields: "MF}' datafile
number of     fields     8
number of     fields     8
number of     fields     8
number of     fields     8
number of     fields     8
number of     fields     8
number of     fields     8
number of     fields     8
number of     fields     8
```

설명

매 레코드에는 8개의 마당이 있다. 내부awk변수 NF는 마당의 번호를 보존하며 매 레코드에 대하여 재설정된다.

% cat datafile

northwest	NW	Joel Craig	3.0	.98	3	4
western	WE	Sharon Kelly	5.3	.97	5	23
southwest	SW	May Chin	2.7	.8	2	18
southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Derek Johnson	4.0	.7	4	17
eastern	EA	Susan Beal	4.4	.84	5	20
northeast	NE	TJ Nichols	5.1	.94	3	13
north	NO	Val hultz	4.5	.89	5	9
central	CY	Sheri Watson	5.7	.94	5	13

3. 패턴과 동작조합에서 정규식

실례 5-38

nawk '/northest/{print $3,$2}' datafile

TJ NE

설명

레코드가 패턴 northeast를 포함하면 세번째 마당과 그뒤의 두번째 마당이 인쇄된다.

실례 5-39

nawk ' /E/ ' datafile

western	*WE*	*Sharon Kelly*	*5.3*	*.97*	*5*	*23*
southeast	*SE*	*Derek Johnson*	*4.0*	*.7*	*4*	*17*
eastern	*EA*	*Susan Beal*	*4.4*	*.84*	*5*	*20*
northeast	*NE*	*TJ Nichols*	*5.1*	*.94*	*3*	*13*

설명

레코드가 E를 포함하면 전체 레코드가 인쇄된다.

실례 5-40

nawk -/^ns]/(print $1}' datafile

northwest

southwest

southern
southeast
northeast
north

설명

레코드가 n이나 s로 시작하면 첫번째 마당이 인쇄된다.

실례 5-41

nawk ' $5 ~/ \ . [7-9]+/ '   datafile

southwest	**SW**	**Chris Foster**	**2.7**	**.8**	**2**	**18**
central	CT	Sheri Watson	5.7	.94	5	13

설명

다섯번째 마당($5)이 하나의 점과 그뒤에 하나이상의 7과 9사이의 수자를 포함하면 레코드가 인쇄된다.

실례 5-42

nawk -$2 !- /E/(print $1, $2)' datafila
northwest NW
soutbwest SW
southern SO
north NO
central CT

설명

두번째 마당이 패턴 E를 포함하지 않으면 첫번째 마당과 그뒤에 오는 두번째 마당($1, $2)이 인쇄된다.

% cat datafile

northwest	NW	Joel Craig	3.0	.98	3	4
western	WE	Sharon Kelly	5.3	.97	5	23
southwest	SW	May Chin	2.7	.8	2	18
southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Derek Johnson	4.0	.7	4	17
eastern	EA	Susan Beal	4.4	.84	5	20
northeast	NE	TJ Nichols	5.1	.94	3	13
north	NO	Val hultz	4.5	.89	5	9
central	CY	Sheri Watson	5.7	.94	5	13

실례 5-43

nawk '$3 - /*Joel/{print $3 " is a nice guy."}- datafile

Joel is a nice guy.

설명

세번째 마당($3)이 패턴 Joel로 시작된다면 세번째 마당과 그뒤에 오는 문자렬 is a nice guy가 인쇄된다. 공백을 인쇄하려면 문자렬안에 포함시켜야 한다.

실례 5-44

nawk "$8 - /[0-91[0-9]$/{print $8}' datafile

23
18
15
17
20
13
13

설명

여덟번째 마당이 두 수자로 끝나면 인쇄된다.

실례 5-45

nawk '$4 ~ /Chin$/{print "The price is $" $8 "."}'datafile

The price is $15.

설명

네번째 마당($4)이 chin으로 끝나면 2중인용부호안의 문자렬("The price is $")과 여덟번째 마당 $8, 그리고 점을 포함하는 문자렬이 인쇄된다.

실례 5-46

nawk '/TJ/{print $0}' ***datafile***

northeast NE TJ Nichols 5.1 .94 3 13

설명

레코드가 패턴 TJ를 가지고 있으면 $0(레코드)이 인쇄된다.

4. 입력마당분리기호

% cat datafile

northwest	NW	Joel Craig	3.0	.98	3	4
western	WE	Sharon Kelly	5.3	.97	5	23
southwest	SW	May Chin	2.7	.8	2	18
southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Derek Johnson	4.0	.7	4	17
eastern	EA	Susan Beal	4.4	.84	5	20
northeast	NE	TJ Nichols	5.1	.94	3	13
north	NO	Val hultz	4.5	.89	5	9
central	CY	Sheri Watson	5.7	.94	5	13

실례 5-47

Nawk '{print $1}' datdfile2
Joel
Sharon
Chris
May
Derek
Susan
TJ
Val
Sheri

설명

기정입력마당분리기호는 공백이다. 첫번째 마당($1)이 인쇄된다.

실례 5-48

nawk -F: '{print $1}' datafile2
Joel Craig
Sharon Kelly
Chris Foster
<more output here>
Val Shultz Sheri Watson

설명

-F추가선택은 두점을 입력마당분리기호로 한다. 첫 마당($1)이 인쇄된다.

실례 5-49

nawk '{print "number of fields: "HP}' datafile2
Number of fields: 2

Number of fields: 2

Number of fields: 2

<more of the same output here>

Number of fields: 2

Number of fields: 2

설명

마당분리기호가 기정값(공백)이므로 매 레코드에 대한 마당의 개수는 2이다. 공백만이 첫번째와 마지막마당사이에 놓인다.

실례 5-50

nawk -F: ' {print " number of fields: " NF} ' datafile2

Number of fields: 7
Number of fields: 7
Number of fields: 7
<more of the same output here>
Number of fields: 7
Number of fields: 7

설명

마당분리기호가 두점이므로 매 레코드에서 마당수는 7이다.

실례 5-51

nawk -F"[:]" '{print $1, $2}' datafile2

설명

다중마당분리기호들은 nawk를 리용하여 정규식으로 지정될수 있다. 공백이나 두점은 마당분리기호로 지정된다. 첫번째와 두번째 마당($1, $2)이 인쇄된다.

% cat datafile

northwest	NW	Joel Craig	3.0	.98	3	4
western	WE	Sharon Kelly	5.3	.97	5	23
southwest	SW	May Chin	2.7	.8	2	18

southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Derek Johnson	4.0	.7	4	17
eastern	EA	Susan Beal	4.4	.84	5	20
northeast	NE	TJ Nichols	5.1	.94	3	13
north	NO	Val hultz	4.5	.89	5	9
central	CY	Sheri Watson	5.7	.94	5	13

5. awk 스크립트작성

실례 5-52

cat nawk.sc1

 # This is a comment

 # This is my first awk script

1. /"north/{print $1, $2, $3}
2. /^south/{print "The " $1 " district."}

nawk -f nawk.sci datafile

3. *northwest NW Joel*

The southwest district.

The southern district.

The southeast district.

Northeast NE TJ

North NO Val

설명

1. 레코드가 패턴 north로 시작하면 레코드의 첫번째, 두번째, 세번째 마당($1, $2, $3)이 인쇄된다.
2. 레코드가 패턴 south로 시작하면 문자렬 The와 그뒤에 있는 첫번째 마당($1)의 값, 그리고 문자렬 district가 인쇄된다.
3. –f추가선택은 awk스크립트파일의 이름앞에 있고 파일이름뒤에 처리되는 입력파일이 있다.

UNIX도구들에 대한 련습

련습문제 3: awk련습

Mike Harrington:(510)548-1278:250:100:175
Christian Dobbins:(408) 538-2358:155:90:201
Susan Dalsass:(206 654-6279:250:60:50
Archie McNichol:(206)548-1348:250:100:175
Jody Savage :(206) 548-1278:15:188:150
Guy Quigley:(916) 343-6410:250:100:175
Dan Savage:(406) 298-7744:450:300:275
Nancy McNeil:(206) 548-1278:250:80:75
John Goldenrod:(916 348-4278:250:100:175
Chet Main:(510) 538-5258:50:95:135
Tom Savage:(408) 926-3456:250:168:200
Elizabeth Stachelin:(916) 440-1763:175:75:300

우의 자료기지는 이름과 전화번호, 세번째 달에 걸쳐 진행한 현금기부금을 포함한다.

1. 전화번호를 모두 인쇄하시오.
2. Dan으로 전화번호를 인쇄하시오.
3. Susan의 이름과 전화번호를 인쇄하시오.
4. D로 시작하는 이름들을 모두 인쇄하시오.
5. C 혹은 E로 시작하는 첫번째 이름들을 모두 인쇄하시오.
6. 4개의 문자들로 되여 있는 첫 이름들을 모두 인쇄하시오.
7. 916지역코드에서 첫 이름을 모두 인쇄하시오.
8. Mike의 기부금을 인쇄하시오. 매개 값은 앞에 화페문자를 달고 인쇄하여야 한다. 실례로 $250$100$175이다.
9. 첫 이름과 반점다음에 마지막이름들을 인쇄하시오.
10. Fact라는 이름으로 awk스크립트를 쓰시오.

   ㄱ. Savarge의 전체 이름들과 전화번호들을 인쇄하시오.
   ㄴ. Chet의 기부금을 인쇄하시오.
   ㄷ. 첫 달에 $250을 기부한 모든 사람들을 인쇄하시오.

제 6 장. awk편의프로그람: awk프로그람작성구조

제 1 절. 비교식

비교식들은 조건이 참일 때 동작이 수행되는 행들을 대조한다. 이 식들은 관계연산자를 사용하며 수자나 문자렬을 비교하기 위해 사용된다. 표 6-1은 관계연산자목록을 보여 주었다. 표현의 값은 그것이 참으로 될 때 1, 거짓일 때 0이다.

1. 관계연산자

표 6-1. 관계연산자

연산자	의 미	실 례
<	작다.	x<y
<=	작거나 같다.	x<=y
==	같다.	x==y
!=	같지 않다.	x!=y
>=	크거나 같다.	x>=y
>	크다.	x>y
~	정규식을 일치시킨다.	x~/y/
!~	정규식을 일치시키지 않는다.	x!~/y/

실례 6-1

(The Database)
% cat employees

Tom Jones 4423 5/12/66 543354
Mary Adams 5346 11/4/63 28765
Sally Black 1654 9/23/44 650000
Billy Black 1683 9/23/44 336500

(The Command Line)

1. **% nawk -$3 == 5346' employees**

   Mary Adams 5346 11/4/63 28765

2. **% nawk "$3 > 5000{print $1}' employees**
   Mary

3. **% nawk '$2 - /Adam/' employees**
   Mary Adams 5346 11/4/63 28765

4. **% nawk '$2 !- /Adam/' employees**

Tom Jones	*4423*	*5/12/66*	*543354*
Sally Black	*1654*	*9/23/44*	*650000*
Billy Black	*1683*	*9/23/44*	*336500*

설명

1. 세번째 마당이 5346과 같으면 조건은 참이고 awk는 기정동작 즉 그 행을 인쇄한다. if조건이 포함되면 그것은 조건패턴검사이다.
2. 세번째 마당이 5000보다 더 크면 awk는 첫 마당을 인쇄한다.
3. 두번째 마당이 정규식 Adam과 일치하면 그 레코드가 인쇄된다.
4. 두번째 마당이 정규식 Adam과 일치되지 않으면 그 레코드가 인쇄된다. 표현이 수값이고 수자비교를 요구하는 연산자를 가진 문자렬값과 비교되면 문자렬값이 수값으로 변환된다. 연산자가 문자렬값을 요구하면 수값은 문자렬값으로 변환된다.

2. 조건식

조건식은 2개의 기호 즉 물음표 ?와 두점 :을 리용하여 표현들을 평가한다. if/else명령문과 같은 결과를 얻기 위한 간단한 방법이다. 일반적인 형식은 다음과 같다.

형식

conditional expression1 ? expression2 : expression3

이렇게 하면 그 결과는 아래의 if/else명령문을 리용할 때와 같다(if/else구조에 대해서는 후에 구체적으로 론의한다.).

```
{
if (expression1)
        expression2
else
        expression3
}
```

실례 6-2

nawk '{max=($l > $2) ? $1 : $2; print max}' filename

설명

첫 마당이 두번째 마당보다 더 크면 물음표의 다음에 있는 식의 값은 max에 대입되고 그렇지 않으면 옹근두점뒤에 있는 식의 값이 max에 대입된다.

이것은 다음것과 비교된다.

```
if($1>$2)
      max=$1
```

```
        else
                max=$2
```

3. 계산

계산은 패턴안에서 수행될수 있다. awk는 류동소수점의 모든 산수연산을 진행한다. 산수연산자들을 표 6-2에 보여 주었다.

표 6-2. 산수연산자

연산자	의 미	실 례
+	더하기	x+y
–	덜기	x-y
*	곱하기	x*y
/	나누기	x/y
%	나머지	x%y
^	제곱	x^y

실례 6-3

```
nawk '$3  *  $4 > 500'   filename
```

설명

awk는 세번째 마당($3)과 네번째 마당($4)을 곱하고 결과가 500보다 크면 그 행들을 표시한다(filename이 입력을 포함하고 있는 파일이라고 가정한다.).

4. 합성패턴

합성패턴은 론리연산자(표 6-3)를 가진 패턴들을 조합하는 표현들이다. 표현은 왼쪽에서부터 오른쪽으로 평가된다.

표 6-3. 론리연산자

연산자	의 미	실 례
&&	론리곱하기	a && b
||	론리더하기	a || b
!	부 정	!a

실례 6-4

```
nawk '$2 > 5 && <= 15' filename
```

설명

awk는 2개의 조건 즉 두번째 마당($2)이 5보다 더 크고 15보다 같거나 작은 그러한 조건과 일치하는 행을 표시한다. &&연산자에서는 두 조건들이 참이 되여야 한다(filename은 입력을 포함하고 있는 파일이라고 가정한다.).

실례 6-5

```
nawk '$3 == 100 || $4 > 50' filename
```

설명

awk는 조건들중 어느 하나라도 일치되는 그러한 행 다시말하여 세번째 마당이 100과 같거나 네번째 마당이 50보다 더 큰 행들을 표시한다.

||연산자에서는 조건들중의 어느 하나만이라도 참이여야 한다(filename은 입력을 포함하고 있는 파일이라고 가정한다.).

실례 6-6

```
nawk '!($2 < 100 && $3 <20 )' filename
```

설명

두 조건이 참이면 awk는 표현식을 부정하고 그 행들을 표시한다.

그러므로 표시되는 행들에서 하나 또는 2개의 조건이 거짓으로 된다.

부정연산자 !는 표현식의 조건이 참(또는 거짓)일 때 not가 그것을 거짓(또는 참)으로 되게 하기 위해 조건식의 결과를 부정한다(fileaname은 입력을 가진 파일이라고 가정한다.).

5. 범위패턴

범위패턴은 첫번째로 나타나는 첫번째 패턴과 두번째 패턴사이를 대조한 다음 두번째로 나타나는 첫번째 패턴과 두번째 패턴사이를 대조하는 식으로 계속한다. 첫번째 패턴이 일치되고 두번째 패턴이 발견되지 않으면 awk는 파일의 끝까지 모든 행들을 표시한다.

실례 6-7

```
nawk '/Tom/,/Suzanne/' filename
```

설명

awk는 포괄적으로 Tom의 첫 발생과 Suzanne의 첫 발생사이에 있는 모든 행들을 표시한다. Suzanne를 찾지 못하면 awk는 파일의 끝까지 모든 행들을 계속 처리한다. 만일 Tom과 Suzanne사이에 있는것이 인쇄된후 Tom이 다시 나타나면 awk는 또 다른 Suzanne이 발견될 때까지 혹은 파일의 끝까지 행들을 표시하기 시작한다.

6. 자료확인프로그람

지금까지 론의된 awk지령을 사용하여 다음의 통과암호검사프로그람이 파일에서 자료를 어떻게 확인하는가를 설명한다.

실례 6-8

(The Password Database)

1. % **cat /etc/passwd**

```
tooth:pwHfud.0.eC9sM:476:40:Contract Admin. :/home/rickenbacker/tooth:/bin/csh
lisam:9JY70uS2f31HY:4467:40:Lisa M. Spencer:/home/fortunel/lisam:/bin/csh
goode:v7Ww.nWJCeSIQ:32555:60:Goodwill Guest User:/usr/goodwill:/bin/csh
bonzo:eTZbu6M2jM7VA:5101:911:SSTOOL Log account :/home/sun4/bonzo: /bin/csh
info :mKZsrioPtW9hA: 611:41: Terri Stern: /home/chewie/info: /bin/csh
cnc:INHVqVjlbVv2:10209: 41: Charles Camell:/home/christine/cnc : /bin/csh
bee:*: 347:40: Contract Temp. :/home/chanelS/'bee: /bin/csh
friedman:oyuIiKoFTVOTE:3561:50:JayFriedman:/home/ibanez/friedman:/bin/csh
chambers:Rw7Rlk77yUY4:592:40:Carol:Chambers:/usr/callisto2/chambers: bin/csh
gregc: nkLulOg: 7777:30: Greg Champlin FE Chicago
ramona:gbDQLdDBeRc46:16660:68:RamonaLeininge:MWA :CustomerService Rep:/
home / forsh:
```

(The *Awk* Commands)

```
2.  % cat /etc/passwd | nawk –F:
3.  NF != 7{\
4.  printf("line %d, does not have 7 fields: %s\ n',NR, $0) ) \
5.  $1 !-/ [A-Za-zO-9]/{printfC'line %d, nonalphanumeric user id: tsNn",TIR,$0.}.} \
6.  $2 == "*" (printfC'line %d, no password: %s".n" ,NR, $0)} '
```

(The Output)

line 7, no password: bee: *: 347: 40 .*-Contract TeJnp. :*home/chanel5/bee : /bin/csh
line 10, *does not have 7 fields:* gregc :nk2EYi7kLulOg: 7777:30:Greg Champlin
FE Chicago
line 11, does'not *hav^* 7fields: ramona :gbDQLdDBeRc46:16660: 68: Ramona
Leininger MWA Customisr Service Rep:/home/forsh:

설명

1. /etc/passwd파일의 내용이 표시된다.
2. cat프로그람은 awk에 자기의 출력을 전송한다. awk의 마당분리기호는 두점이다.
3. 마당의 번호(NF)가 7과 같지 않을 때 그다음의 동작블로크가 실행된다.
4. printf함수는 문자렬 Line(number), does not have 7 files:를 인쇄한다. 그뒤에 현재레코드번호(NR)와 레코드자체($0)를 인쇄한다.
5. 첫번째 마당($1)이 그 어떤 수자와 자모문자들도 포함하지 않으면 printf함

수는 문자렬 nonalphanumeric user id:와 그뒤에 레코드번호와 레코드를 인쇄한다.

6. 두번째 마당($2)이 별표와 같으면 문자렬 no passwd:는 인쇄되고 그뒤에 레코드번호와 레코드 그자체가 인쇄된다.

제 2 절. 개괄

1. 동등성검사

```
% cat datafile
northwest    NW    Joel Craig        3.0    .98    3    4
western      WE    Sharon Kelly      5.3    .97    5    23
southwest    SW    Chris Foster      2.7    .8     2    18
southern     SO    May Chin          5.1    .95    4    15
southeast    SE    Derek Johnson     4.0    .7     4    17
eastern      EA    Susan Beal        4.4    .84    5    20
northeast    NE    TJ Nichols        5.1    .94    3    13
north        NO    Val hultz         4.5    .89    5     9
central      CY    Sheri Watson      5.7    .94    5    13
```

실례 6-9

```
nawk '$7    == 5' datafile
western    WE    Sharon Kelly     5.3    .97    5    23
eastern    EA    Susan Beal       4.4    .84    5    20
north      NO    Val Shultz       4.5    .89    5    9
central    CT    Sheri Watson     5.7    .94    5    13
```

설명

일곱번째 마당이($7) 5와 같으면 그 레코드가 인쇄된다.

실례 6-10

```
nawk '$2 = = “CT”{print $1, $2}' datafile
central        CT
```

설명

두번째 마당이 CT문자렬과 같으면 마당 1과 2($1, $2)가 인쇄된다. 문자렬들은 인용부호안에 넣어야 한다.

```
% cat datafile
northwest    NW    Joel Craig       3.0    .98    3    4
western      WE    Sharon Kelly     5.3    .97    5    23
southwest    SW    Chris Foster     2.7    .8     2    18
southern     SO    May Chin         5.1    .95    4    15
southeast    SE    Derek Johnson    4.0    .7     4    17
eastern      EA    Susan Beal       4.4    .84    5    20
northeast    NE    TJ Nichols       5.1    .94    3    13
north        NO    Val hultz        4.5    .89    5    9
central      CY    Sheri Watson     5.7    .94    5    13
```

2. 관계연산자

실례 6-11

```
nawk '$7 != 5' datafile
northwest    NW    Joel Cralg       3.0    98     3    4
southwest    SW    Chris Foster     2.7    .8     2    18
southern     SO    May Chin         5.1    .95    4    15
southeast    SE    Derek Johnson    4.0    .7     4    17
northeast    NE    TJ Nicholas      5.1    .94    3    13
```

설명

일곱번째 마당($7)이 수자 5와 같지 않으면 그 레코드를 인쇄한다.

실례 6-12

```
nawk '$7 < 5 {print $4, $7}' datafile
Craig 3
Foster 2
Chin 4
Johnson 4
Nichols 3
```

설명

일곱번째 마당($7)이 5보다 작으면 마당 4와 7을 인쇄한다.

실례 6-13

```
nawk  '$6 > .9 {print  $1,  $6}' datafile
```

```
northwest .98
western .97
southern .95
northeast .94
central .94
```

설명

여섯번째 마당($6)이 9보다 크면 마당 1과 6이 인쇄된다.

실례 6-14

```
nawk '$8 <= 17 {print $8}' datafile
4
15
17
13
9
13
```

설명

여덟번째 마당($8)이 17과 같거나 작으면 그 마당을 인쇄한다.

실례 6-15

```
nawk '$8 >= 17 {print $8}' datafile
23
18
17
20
```

설명

여덟번째 마당이 17이상이면 그 마당을 인쇄한다.

```
% cat datafile
northwest   NW   Joel Craig       3.0   .98   3    4
western     WE   Sharon Kelly     5.3   .97   5   23
southwest   SW   May Chin         2.7   .8    2   18
southern    SO   Suan Chin        5.1   .95   4   15
southeast   SE   Derek Johnson    4.0   .7    4   17
eastern     EA   Susan Beal       4.4   .84   5   20
northeast   NE   TJ Nichols       5.1   .94   3   13
north       NO   Val hultz        4.5   .89   5    9
central     CY   Sheri Watson     5.7   .94   5   13
```

3. 론리연산자

실례 6-16

nawk '$8 > 10 && $8 < 17' datafile

```
southern    SO    May Chin        5.1    .95    4    15
northeast   NE    TJ Nichols      5.1    .94    3    13
central     CT    Sheri Watson    5.7    .94    5    13
```

설명

여덟번째 마당이 10보다 크고 17보다 작으면 그 레코드를 인쇄한다. 레코드는 두 표현들이 다 참일 때만 인쇄된다.

실례 6-17

nawk '$2 == "MW" | | $1 - /south/{print $1, $2}' datafile

```
northwest NW
southwest SW
southern SO
southeast SE
```

설명

두번째 마당($2)이 문자렬 NW와 같거나 첫 마당($1)이 패턴 south를 가지고 있으면 첫번째, 두번째 마당($1, $2)이 인쇄된다. 레코드는 두 표현이 다 참일 때만 인쇄된다.

4. 론리부정연산자

실례 6-18

nawk '!($8 == 13){print $8}' datafile

```
4
23
18
15
17
20
9
```

설명

여덟번째 마당($8)이 13과 같으면 부정 !는 표현식을 부정하고 여덟번째 마당($8)을 인쇄한다. !는 부정연산자이다.

5. 산수연산자

실례 6-19

```
Nawk '/southern/{print $5 + 10}' datafile
15.1
```

설명

레코드가 정규식 southern을 포함하면 10이 다섯번째 마당($5)의 값에 더해져 인쇄된다. 그 수자는 류동소수점수로 인쇄된다는것을 주의하여야 한다.

실례 6-20

```
nawk '/southern/{print $8 + 10}' datafile
25
```

설명

레코드가 정규식 southern을 가지고 있으면 10이 여덟번째 마당($8)값에 더해 져 인쇄된다. 그 수는 10진수로 인쇄된다는것을 주의하여야 한다.

```
% cat datafile
Northwest   NW  Joel Craig      3.0  .98  3   4
western     WE  Sharon Kelly    5.3  .97  5  23
southwest   SW  Chris Foster    2.7  .8   2  18
southern    SO  May Chin        5.1  .95  4  15
southeast   SE  Derek Johnson   4.0  .7   4  17
eastern     EA  Susan Beal      4.4  .84  5  20
northeast   NE  TJ Nichols      5.1  .94  3  13
north       NO  Val hultz       4.5  .89  5   9
central     CY  Sheri Watson    5.7  .94  5  13
```

실례 6-21

```
nawk '/southern/{print $5 + 10.56}' datdfile
15.66
```

설명

레코드가 정규식 southern을 포함하면 10.56이 다섯번째 마당($5)의 값에 추

가되여 인쇄된다.

실례 6-22

nawk '/southern/{print $8 –10}' datdfile
5

설명

레코드가 정규식 southern을 포함하면 여덟번째 마당($8)값에서 10을 덜어 인쇄된다.

실례 6-23

nawk '/southern/{print $8 /2}' datafile
7.5

설명

레코드가 정규식 southern을 포함하면 여덟번째 마당($8)의 값이 2로 나누어져 인쇄된다.

실례 6-24

nawk '/ northest/{print $8 / 3}' datafile
4.33333

설명

레코드가 정규식 northern을 포함하면 여덟번째 마당($8)을 3으로 나누어 인쇄한다. 정확도는 소수점아래 5자리까지이다.

실례 6-25

nawk '/southern/{print $8 * 2}' datafile
30

설명

레코드가 정규식 southern을 가지면 여덟번째 마당($8)에 2가 곱해 져 인쇄된다.

실례 6-26

nawk '/northest/{print $8 % 3}' datafile
1

설명

레코드가 정규식 northeast를 가지면 여덟번째 마당($8)을 3으로 나눈 나머지

가 인쇄된다.

실례 6-27

```
nawk '$3 ~ /^Susan/\
{print "Percentage: "$6 + .2 " Volume: "$8}' datafile
Percentage: 1.04 Volume:20
```

설명

세번째 마당($3)이 정규식 Susan으로 시작되면 print함수는 계산결과와 2중인용부호안에 있는 문자렬을 인쇄한다.

% cat datafile

northwest	NW	Joel Craig	3.0	.98	3	4
western	WE	Sharon Kelly	5.3	.97	5	23
southwest	SW	Chris Foster	2.7	.8	2	18
southern	SO	May Chin	5.1	.95	4	15
southeast	SE	Derek Johnson	4.0	.7	4	17
eastern	EA	Susan Beal	4.4	.84	5	20
northeast	NE	TJ Nichols	5.1	.94	3	13
north	NO	Val hultz	4.5	.89	5	9
central	CY	Sheri Watson	5.7	.94	5	13

6. 범위연산자

실례 6-28

```
nawk '/'western/,/*eastern/' datafile
western     WE   Sharon Kelly     5.3   .97   5   23
southwest   SW   Chris Foster     2.7   .8    2   18
southern    SO   May Chin         5.1   .95   4   15
southeast   SE   Derek Johns on   4.0   .7    4   17
eastern     EA   Susan Beal       4.4   .84   5   20
```

설명

정규식 western으로 시작하는 범위안의 모든 레코드들은 정규식 eastern으로 시작되는 레코드를 찾을 때까지 인쇄된다. 레코드들은 패턴 western이 발견되면 다시 인쇄되기 시작하여 eastern이나 파일의 끝을 발견할 때까지 계속 인쇄된다.

7. 조건연산자

실례 6-29

```
nawk '{print ($7 > 4 ? "high "$7 ; "low "$7)}-' datafile
low 3
high 5
low 2
low 4
low 4
high 5
low 3
high 5
high 5
```

설명

일곱번째 마당($7)이 4보다 크면 print함수는 물음표뒤에 표현값(문자렬 high와 일곱번째 마당)을 취하고 4보다 크지 않으면 print함수는 두점뒤에 표현값(문자렬 low와 일곱번째 마당값)을 취한다.

8. 대입연산자

실례 6-30

```
nawk '$3 == "chris"{ $3 = "Christian"; print}' datafile
southwest SW Christian Foster 2.7 .82 18
```

설명

세번째 마당($3)이 문자렬 chris와 같으면 동작은 세번째 마당($3)에 christian을 대입하고 그 레코드를 인쇄한다. 2중같기기호(==)는 연산항들이 같은가를 검사하고 단일같기기호(=)는 대입연산에 리용된다.

실례 6-31

```
nawk '/Dreck/{$8 += 12; print $8}' datafile
29
```

설명

정규식 Derek를 찾으면 수 12에 여덟번째 마당($8)을 더하고 $8에 대입한후 그 값을 인쇄한다. 이것을 다르게 표현할수도 있다. 즉 $8=$8+12이다.

실례 6-32

```
nawk '{$7 %= 3; print $7}' datafile
0
2
```

2
1
1
2
0
2
2

설명

매 레코드에 대하여 일곱번째 마당($7)을 3으로 나눈 나머지를 $7에 대입하고 인쇄한다.

UNIX TOOLS의 련습

련습문제 4: awk 련습

Mike Harrington:(510)548-1278:250:100:175
Christian Dobbins:(408) 538-2358:155:90:201
Susan Dalsass:(206 654-6279:250:60:50
Archie McNichol:(206)548-1348:250:100:175
Jody Savage :(206) 548-1278:15:188:150
Guy Quigley:(916) 343-6410:250:100:175
Dan Savage:(406) 298-7744:450:300:275
Nancy McNeil:(206) 548-1278:250:80:75
John Goldenrod:(916 348-4278:250:100:175
Chet Main:(510) 538-5258:50:95:135
Tom Savage:(408) 926-3456:250:168:200
Elizabeth Stachelin:(916) 440-1763:175:75:300

(CD에 있는 lab4.data라는 자료기지에 있다.)

우의 자료기지는 이름, 전화번호, 세번째 달에 걸치는 현금기부금을 포함한다.

1. 첫 달에 100$이상 기부한 사람의 첫번째, 두번째 이름을 인쇄하시오.
2. 첫 달에 60$이하 기부한 사람의 이름과 전화번호를 인쇄하시오.
3. 세번째 달에 90$부터 150$까지 기부한 사람들을 인쇄하시오.
4. 세 달에 걸쳐 800$보다 더 많이 기부한 사람들을 인쇄하시오.
5. 달마다 평균 150$보다 더 많이 기부한 사람의 이름과 전화번호를 인쇄하시오.
6. 916지역코드에 없는 사람들의 첫 이름을 인쇄하시오.
7. 매 레코드앞에 레코드번호를 인쇄하시오.
8. 매 사람의 이름과 총 기부금을 인쇄하시오.
9. Elizabeth의 두번째 기부금에 10$를 더하시오.

Money McNeil의 이름을 Louise MCInnes로 바꾸시오.

제 7 장. awk편의프로그람: awk프로그람작성법

제 1 절. 변수

1. 수자와 문자렬상수

수자상수는 243과 같은 옹근수와 3.14와 같은 류동소수점수 혹은 723E-1이나 3.4e7과 같은 파학적표기법으로 표기한 수자들로 표현할수 있다. Hello World를 비롯한 문자렬들은 인용부호안에 넣는다.

초기화와 형일치 awk프로그람에서 변수를 선언하면 그것이 존재하게 된다. 변수는 문자렬, 수자 혹은 그 조합들이 될수 있다. 변수는 설정될 때 갈기기호의 오른쪽에 있는 표현형으로 된다. 초기화되지 않은 변수들은 그가 사용되는 문맥에 따라 값 0 혹은 값 " "을 가진다.

name "Nancy" name is a string
x++ ***x*** *is a number*
x *is initialized to zero and incremented by 1*
number number is a number

문자렬을 수자처럼 취급하려면

name+0

수자를 문자렬처럼 취급하려면

number " "

로 선언해야 한다.

split함수에 의해 작성된 모든 마당들과 배렬요소들은 수값만을 포함하지 않으면 문자렬로 고찰한다. 마당 혹은 배렬요소가 빈 값이면 그것은 빈 문자렬값을 가진다. 빈 행도 빈 문자렬로 본다.

2. 사용자정의변수

사용자정의변수들은 글자, 수자, 밑줄로 구성되며 수자로는 시작할수 없다. awk에서 변수는 선언되지 않는다. awk는 표현에서 변수의 문맥에 따라 자료형을 추측한다. 만일 변수가 초기화되지 않으면 awk는 문자렬변수를 빈 값으로 초기화하고 수값변수들은 0으로 초기화한다. 필요하면 awk는 문자렬변수를 수값변수로 또는 그 반대로 바꿀수 있다. 변수들에는 awk의 대입연산자로 변수들의 값을 대입한다. 대입연산자를 표 7-1에서 보여 주었다.

표 7-1. 대입연산자

연산자	의 미	동등관계
=	a=5	a=5
+=	a=a+5	a+=5
-=	a=a-5	a-=5
*=	a=a*5	a*=5
/=	a=a/5	a/=5
%=	a=a%5	a%=5
^=	a=a^5	a^=5

가장 간단한 대입은 표현결과를 취하여 변수에 그것을 대입하는것이다.

형식

variable = expression

실례 7-1

```
% nawk '$1 -  /Tom/ {wage = $2  *  $3; print wage}'  filename
```

설명

awk는 Tom의 첫 마당을 검사하고 일치하면 두번째 마당값에 세번째 마당값을 곱하여 결과를 사용자정의변수 Wage에 대입한다. 곱하기연산이 산수연산이므로 awk는 0의 초기화값을 Wage에 대입한다(%는 UNIX재촉문이고 filename은 입력파일이다.).

증가와 감소연산자 연산수에 1을 더하려면 증가연산자를 사용한다. 표현 x++는 x=x+1과 동등하다. 이와 류사하게 감소연산자는 연산수에서 1을 던다. 표현 x--는 x=x-1과 동등하다. 이 표현식은 계수기를 간단히 증가, 감소하려 할 때 순환연산에서 쓸모가 있다. ++x에서와 같이 연산자앞에 또는 x++에서와 같이 연산자뒤에 증가, 감소연산자들을 사용할수 있다. 이 표현들을 대입명령문에 사용하면 그 배치에 따라 연산결과가 달라 진다.

```
{x=1;y=x++;print x,y};
```

여기서 ++는 처리후(post)증가연산자라고 부르며 y에 1의 값이 대입되고 그다음에 x가 1만큼 증가하여 y는 1, x는 2와 같게 된다.

```
{x=1;y=++x;print x,y};
```

여기서 ++는 처리전(pre)증가연산자이며 x가 먼저 증가하고 2의 값이 y에 대입되며 이 명령문을 끝내면 y는 2이고 x는 2이다.

지령행에서 사용자정의변수 지령행에서 하나의 변수는 하나의 값을 대입 받을수 있으며 awk스크립트에 넘겨 질수 있다. 인수를 처리하는데서 구체적인것은 165페지의

《지령인수처리(nawk)》보고 ARGV에 대해서는 지령행인수들의 배렬을 보면 된다.

실례 7-2

```
nawk -F ; -f   awkscript     month=4   year=2001 filename
```

설명

사용자정의변수 month와 year에는 각각 4와 2001의 값이 대입된다. awk스크립트에서는 이 변수들이 마치 스크립트에서 작성된것처럼 사용할수 있다.

주의: filename이 인수앞에 있으면 변수들은 BEGIN명령문에서 쓸수 없다(135페지에서 《BEGIN패턴》을 보면 된다.).

-v추가선택(nawk) nawk에 의해 제공되는 -v추가선택은 지령행인수들이 BEGIN명령문안에서 처리되도록 한다. 지령행에 넘겨 지는 매 인수들에 대해서는 그앞에 -v추가선택이 있어야 한다.

마당변수 마당변수들은 마당들을 참고할 때를 제외하고는 사용자변수와 같이 사용된다. 새로운 마당들은 대입에 의해 작성할수 있다. 참조는 되고 값을 가지지 않는 마당값은 빈값문자렬로 대입된다. 마당값이 변화되면 $0변수는 OFS의 현재값을 마당분리기호로 리용하여 다시 계산된다. 허용되는 마당수는 보통 100으로 제한된다.

실례 7-3

```
nawk ‘ { $5 = 1000   *   $3   /   $2 ;   print }’ filename
```

설명

$5가 존재하지 않으면 awk는 그것을 작성하여 거기에 표현 1000*$3의 결과를 대입한다. 다섯번째 마당이 존재하면 거기에 덧쓰기하는 방법으로 결과를 대입한다.

실례 7-4

```
% nawk ‘ $4 == “CA” { $4 = “Cailfornia”; print }’ filename
```

설명

네번째 마당($4)이 문자렬 CA와 같으면 awk는 네번째 마당에 California를 다시 대입한다. 2중인용부호는 반드시 있어야 한다. 이것들이 없으면 문자렬들이 빈값을 초기값으로 가진 사용자정의변수로 된다.

내부변수 내부변수는 대문자이름을 가진다. 내부변수들은 표현에 사용되며 다시 재설정될수 있다. 내부변수목록을 표 7-2에서 보여 주었다.

표 7-2. 내부변수

변수이름	변수내용
ARGC	지령행인수의 번호
ARGV	지령행인수의 배렬

(표계속)

변수이름	변수내용
FILENAME	현재입력파일의 이름
FNR	현재파일에서 레코드수
FS	입력마당분리기호이다(기정값은 공백이다.).
NF	현재레코드에서 마당의 수
NR	레코드의 수
OFMT	수자들의 출력형식
OFS	출력마당분리기호
ORS	출력레코드분리기호
RLENGTH	match함수에 의해 일치되는 문자렬의 길이
RS	입력레코드분리기호
RSTART	Match함수에 의해 일치되는 문자렬의 편차값
UBSEP	부분스크립트분리기호

실례 7-5

(The Employees Database)

% **cat employees2**

Tom Jones:4423:5/12/66:543354 Mary
Adams:5346:11/4/63:28765 Sally
Chang:1654:7/22/54:650000 Mary
Black:1683:9/23/44:336500

(The Command Line)

% **nawk -F: •$1 == "Mary Adams"(print NR, $1, $2, $MF}• employees2**

(The Output)

2 Mary Adams 5346 *28765*

설명

-F추가선택은 마당식별자를 두점으로 설정한다. print함수는 레코드번호, 첫번째, 두번째, 마지막마당($NF)을 인쇄한다.

3. BEGIN패턴

BEGIN패턴은 awk가 입력파일로부터 임의의 행들을 처리하기전에 실행되는 동작블로크앞에 놓인다. 사실 BEGIN블로크는 어떤 입력파일이 없이도 검사될수 있다. 그것은 awk가 BEGIN동작블로크가 끝날 때까지 입력읽기를 시작하지 않기때문이다.

BEGIN동작은 내부변수들인 OFS, RS, FS 등의 값들을 변화시키고 사용자정의변수

들에 초기값을 대입하며 출력부분으로서 머리부나 제목들을 인쇄하기 위해 흔히 사용된다.

실례 7-6

% nawk ‘ BEGIN {FS=“:”;　OFS=“/t; ORS=“\ n\ n”}{print $1, $2, $3} ’ file

설명

입력파일이 처리되기전에 마당분리기호(FS)는 두점으로, 출력마당분리기호(OFS)는 타브로, 출력레코드분리기호(ORS)는 2개의 행바꾸기로 설정된다. 동작블로크에 2개이상의 명령문이 있으면 반두점으로 구분되거나 다른 행들에 배치되여야 한다(쉘이 재촉문상태에 있으면 행바꾸기를 탈퇴시키기 위해 거꿀빗선 \ 을 사용한다.).

실례 7-7

% nawk ' BEGIN {print “ MAKE　YEAR”}'

make year

설명

awk는 MAKE YEAR를 표시한다. print함수는 awk가 입력파일을 열기전에 실행되며 입력파일이 대입되지 않았어도 awk는 MAKE와 YEAR를 인쇄한다. awk스크립트의 오유를 수정할 때에는 프로그람의 나머지부분을 써넣기전에 BEGIN 블로크동작들을 검사할수 있다.

4. END패턴

END패턴은 어떤 입력행과 일치시키는 기능을 수행하지는 않지만 END패턴과 관련된 동작들을 수행한다. END패턴은 입력의 모든 행들이 처리된후에 조종된다.

실례 7-8

% nawk ‘ END {print “The number of records is “ NR }’ filename

The number of records is 4

설명

END블로크는 awk가 파일처리를 끝낸후에 실행된다. NR의 값은 읽은 마지막 레코드번호이다.

실례 7-9

% nawk‘ Mary /{count++}END{print“Mary was found”count“times..”}’\ employees

Mary was found 2 times.

설명

패턴 sun을 포함하는 모든 행에 대하여 count변수의 값은 하나씩 증가된다.

awk가 입력파일을 처리한후에 END블로크는 문자렬 Sun was found 와 count의 값, 문자렬 times를 인쇄한다.

제 2 절. 방향바꾸기와 파이프

1. 출력방향바꾸기

awk내부로부터 UNIX파일까지 출력을 방향바꾸기할 때 쉘방향바꾸기연산자가 사용된다. 파일이름은 2중인용부호안에 놓여야 한다. 기호 >를 사용할 때 파일이 열려 져 끝이 잘리운다. 일단 파일이 열리면 정확이 닫기거나 awk프로그람이 끝날 때까지 열린 상태를 유지한다. 그다음의 print명령들로부터 그 파일에로의 출력은 파일에 덧붙여 진다.

>>기호는 파일열기에는 사용되지만 그것을 지우지는 않는다. 다만 그에 추가시킨다.

실례 7-10

% nawk '$4 > = 70 {print $1. $2 > "passing_file" }' filename

설명

네번째 마당값이 70과 같거나 크면 첫번째, 두번째 마당을 파일 passing_file에로 인쇄된다.

2. 입력방향바꾸기(getline)

getline**함수** getline함수는 표준입력, 파이프 또는 현재 처리되는 파일이 아닌 파일로부터 입력을 읽는데 사용된다. 이 함수는 입력의 다음행을 받아 NF, NR, FNR내부변수들을 설정한다. getline함수는 레코드가 발견되면 1을, EOF(파일의 끝)가 발견되면 0을 되돌려 준다. 파일열기실패와 같은 오유가 있으면 getline함수는 -1의 값을 돌려 준다.

실례 7-11

% nawk ' BEGIN{ "date" | getline d; print d}' filename

The Jan 14 11:24:24 PST 2001

설명

UNIX date지령을 실행하고 출력은 getline에 파이프하고 사용자정의변수 d에 대입한 다음 d를 인쇄한다.

실례 7-12

% nawk 'BEGIN{"date" | getline d; split(d, mon); print mon[2]}'\ filename

Jan

설명

date지령을 실행하고 getline에 출력을 파이프한다. getline함수는 파이프로부터

읽고 사용자정의변수 d에 입력을 기억한다. split함수는 변수 d로부터 mon이라는 배렬을 작성하고 배렬 mon의 두번째 요소를 인쇄한다.

실례 7-13

```
% nawk "BEGIN{while("ls" | getline) print}'
a. out
db
dbook
getdir
file
sortedf
```

설명

ls지령의 출력을 getlite에로 전송한다. 즉 순환의 매 반복에서 getline은 ls로부터 하나이상의 출력행을 읽어 현시장치에 출력한다. 입력파일은 필요 없다. 왜냐하면 BEGIN블로크는 awk가 입력을 열기전에 처리되기때문이다.

실례 7-14

(The Command Line)

```
1. %nawk ' BEGIN{printf "what is yourname?" ; \ getline name<"/dev/tty"}\

2. $1 - name {print " Found " name " on line ", NR "."}\
3. END {print "See ya, " name "."}' filename
```

(The Output)

What is your name? ***Ellie*** < *Waits for input from* user >
Found Elite on line 5.
See ya. Elite.

설명

1. 현시장치에 What is your name?을 인쇄하고 사용자응답을 기다린다. getline함수는 행바꾸기가 입력될 때까지 말단(/dev/tty)으로부터 입력을 받아 사용자정의변수 name에 기억한다.
2. 첫 마당이 name에 대입된 값과 같으면 print함수가 실행된다.
3. END명령문은 See ya를 인쇄하고 그다음에 변수 name에 기억된 값 Ellie가 현시되며 그뒤에 점이 놓인다.

실례 7-15

(The Command Line)

```
% nawk ' BEGIN{while (getline < "/etc/passwd" >0 )lc++; print lc}'\   file
```

(The Output)
16

설명

awk는 /etc/passwd의 파일에서 매행을 읽고 EOF가 탐색될 때까지 lc를 증가시키고 그 값을 인쇄하는데 여기서 lc는 passwd파일에서 행의 수이다.
주의 : getline에 의해 되돌려 지는 값은 파일이 존재하지 않을 때 -1이다. 파일의 끝이 탐색되면 되돌이값은 0, 하나의 행을 읽으면 되돌이값은 1이다. 따라서 지령

while (getline<"/etc/junk")

은 부정값인 –1의 되돌이값이 참조건을 만들기때문에 파일 /etc/junk가 존재하지 않으면 무한순환을 시작한다.

제 3 절. 파이프(pipe)

awk프로그람에서 파이프를 열면 또 다른 파이프를 열기전에 닫아야 한다. 파이프기호의 오른쪽에 있는 지령들은 2중인용부호안에 놓인다. 한번에 하나의 파이프만이 열려 져야 한다.

실례 7-16

(The Database)
% **cat names**
john smith
slice cheba
george goldberg
susan goldberg
tony tram
barbara nguyen
elizabeth lone
dan savage
eliza golcjberg
john goldenrod

(The Command Line)
% **nawk '{print $1, $2 | "sort -r +1 -2 +0 -1 "}' names**

(The Output)
tony tram
john smith
dan savage
barbara nguyen
elizabeth lone
john goldenrod

susan goldberg
george goldberg
eliza goldberg
alice chebe

설명

awk는 print명령의 출력을 입력으로서 UNIX sort지령에 파이프하는데 그것은 두번째 마당을 1차건으로 하고 첫번째 마당을 2차건으로 하여 반대의 순서로 분류한다. UNIX지령은 2중인용부호안에 있어야 한다(부록 1에서 "sort"를 보여 주었다.).

제 4 절. 파일과 파이프의 닫기

awk프로그람에서 파일 혹은 파이프를 다시 읽거나 쓰기 위해 사용하려면 먼저 그것을 닫을 필요가 있다. 왜냐하면 스크립트가 끝날 때까지 열려 져 있기때문이다. 일단 파이프가 열리면 awk가 탈퇴할 때까지 열린상태를 그대로 유지한다. 더우기 END블로크에서 명령문들은 파이프의 영향을 받게 된다. END블로크에서 첫 행은 파이프를 닫는다.

실례 7-17

(In Script)

```
1.  { print $1, $2, $3 | " sort -r +1 -2 +0 -1"}
    END{
2.  close("sort -r +1 -2 +0 -I")
    <rest of statements>   }
```

설명

1. awk는 입력파일로부터 UNIX sort편의프로그람까지 매행을 파이프한다.
2. END블로크가 탐색될 때 파이프가 닫긴다. 2중인용부호안에 있는 문자렬은 파이프가 처음에 열린 파이프문자렬과 같아야 한다.

system함수 내부 system함수는 UNIX체계지령을 인수로 하여 그것을 실행하고 awk프로그람에 탈퇴상태를 되돌려 준다. 이것은 system이라는 c표준서고함수와 류사하다. UNIX지령은 2중인용부호안에 있어야 한다.

형식

```
system( "UNIX Command")
```

실례 7-18

(In Script)

```
    {
1. system ( "cat" $1 )
2. system ( "clear" )
    }
```

설명

1. system함수는 인수로서 UNIX cat지령과 입력파일의 첫번째 마당의 값을 취한다. cat지령은 그의 인수로서 파일이름인 첫번째 마당의 값을 가진다. UNIX쉘은 cat지령을 실행되게 한다.
2. System함수는 인수로서 UNIX clear지령을 가진다. 쉘은 현시장치를 지우기 위해 clear지령을 실행한다.

제 5 절. 개괄

1. 증가, 감소연산자

```
% cat datafile
northwest    NW   Joel Craig       3.0   .98   3    4
western      WE   Sharon Kelly     5.3   .97   5   23
southwest    SW   Chris Foster     2.7   .8    2   18
southern     SO   May Chin         5.1   .95   4   15
southeast    SE   Derek Johnson    4.0   .7    4   17
eastern      EA   Susan Beal       4.4   .84   5   20
northeast    NE   TJ Nichols       5.1   .94   3   13
north        NO   Val Shultz       4.5   .89   5    9
central      CY   Sheri Watson     5.7   .94   5   13
```

실례 7-19

```
% nawk '/*north/{count += 1} print count}' datafile
1
2
3
```

설명

레코드가 정규식 north로 시작하면 사용자정의변수 count가 작성된다. count는 하나 증가되고 그 값이 인쇄된다.

실례 7-20

```
% nawk '/^north/{count++; print count}' datafile
1
2
3
```

설명

자동증가연산자는 사용자정의변수 count를 하나씩 증가시킨다. count값이 인쇄된다.

% cat datafile

northwest	NW	Joel Craig	3.0	.98	3	4
western	WE	Sharon Kelly	5.3	.97	5	23
southwest	SW	Chris Foster	2.7	.8	2	18
southern	SO	May Chin	5.1	.95	4	15
southeast	SE	Derek Johnson	4.0	.7	4	17
eastern	EA	Susan Beal	4.4	.84	5	20
northeast	NE	TJ Nichols	5.1	.94	3	13
north	NO	Val Shultz	4.5	.89	5	9
central	CY	Sheri Watson	5.7	.94	5	13

실례 7-21

```
% nawk '{x = $7--; print "x = "x ", $7 = "$7}' datafile
x = 3, $7=2
x = 5, $7 = 4
x = 2, $7=1
x = 4, $7=3
x = 4, $7=3
x = 5, $7=4
x = 3, $7=2
x = 5, $7=4
x = 5, $7=4
```

설명

일곱번째 마당($7)값이 사용자정의변수 x에 배당된후 자동감소연산자는 일곱번째 마당을 하나씩 감소시킨다. X의 값과 일곱번째 마당값이 인쇄된다.

2. 내부변수

실례 7-22

```
% nawk '^north/ {print "The record number is " NR}・datafile
The record number is 1
The record number is 7
The record number is 8
```

설명

레코드가 정규식 north로 시작되면 문자렬 The record number is와 NR(레코드번호)값이 인쇄된다.

실례 7-23

% nawk '{print NR, $0}' datafile

```
1. northwest  NW  Joel Craig     3.0  98   3  4
2. western    WE  Sharon Kelly   5.3  .97  5  23
3. southwest  SW  Chris Foster   2.7  .8   2  18
4. southern   SO  May Chin       5.1  .95  4  15
5. southeast  SE  Derek Johnson  4.0  .7   4  17
6. eastern    EA  Susan Beal     4.4  .84  5  20
7. northeast  NE  TJ Nichols     5.1  .94  3  13
8. north      NO  Val Shultz     4.5  .89  5  9
9. central    CT  Sheri Watson   5.7  .94  5  13
```

설명

현재레코드번호 NR의 값과 전체 레코드인 $0의 값이 인쇄된다.

실례 7-24

% **nawk 'NR==2,NR==5{print NR,$0}' datafile**

```
2. western    WE  Sharon Kelly   5.3  .97  5  23
3. southwest  SW  Chris Foster   2.7  .8   2  18
4. southern   SO  May Chin       5.1  .95  4  15
5. southeast  SE  Derek Johnson  4.0  .7   4  17
```

설명

NR의 값이 2와 5사이의 범위에 있으면(레코드번호 2~5) 레코드번호(NR)와 그 레코드($0)가 인쇄된다.

실례 7-25

% nawk '^north/(print MR, $1, $2, $MF, RS)' datafile

```
1. northwest  NW  Joel Craig     3.0  98   3  4
7. northeast  NE  TJ Nichols     5.1  .94  3  13
8. north      NO  Val Shultz     4.5  .89  5  9
```

설명

레코드가 정규식 north로 시작되면 레코드번호(NR)뒤에 첫번째, 두번째 마당, 마지막레코드의 값($NF)과 RS의 값(행바꾸기)이 인쇄된다. print함수가 기정

으로 행바꾸기를 발생하므로 RS는 또 다른 행바꾸기를 발생하여 레코드들사이에 2개의 행공간이 있게 된다.

```
% cat datafile
northwest    NW   Joel Craig       3.0   .98   3    4
western      WE   Sharon Kelly     5.3   .97   5   23
southwest    SW   Chris Foster     2.7   .8    2   18
southern     SO   May Chin         5.1   .95   4   15
southeast    SE   Derek Johnson    4.0   .7    4   17
eastern      EA   Susan Beal       4.4   .84   5   20
northeast    NE   TJ Nichols       5.1   .94   3   13
north        NO   Val Shultz       4.5   .89   5    9
central      CY   Sheri Watson     5.7   .94   5   13
```

```
% cat datafile2
Joel Craig:northwest:           NW: 3.0 : .98 : 3 : 4
Sharon Kelly:western:           WE: 5.3 : .97 : 5 : 23
Chris Foster:southwest:         SW: 2.7 : .8  : 2 : 18
May Chin:southern:              SO: 5.1 : .95 : 4 : 15
Derek Johnson:southeast:        SE: 4.0 : .7  : 4 : 17
Susan Beal:eastern:             EA: 4.4 : .84 : 5 : 20
TJ Nichols:northeast:           NE: 5.1 : .94 : 3 : 13
Val hultz:north:                NO: 4.5 : .89 : 5 : 9
Sheri Watson:central:           CY: 5.7 : .94 : 5 : 13
```

실례 7-26

```
% nawk –F == 5{print NF}' datafile2
7
```

설명

마당분리기호는 –F추가선택을 가진 지령행에서 두점으로 설정된다. 만일 레코드번호(NR)가 5이면 마당번호(NF)를 인쇄한다.

실례 7-27

```
% nawk •BEGIN{OFMT=«%.2f;print 1.2456789,12E-2}• datafile2
1.25 0.12
```

설명

OFMT 즉 print함수의 출력형식변수는 류동소수점수가 소수점 두자리의 정확도로 인쇄되도록 설정된다. 123456789와 12E-2은 새 형식으로 인쇄된다.

% cat datafile

northwest	NW	Joel Craig	3.0	.98	3	4
western	WE	Sharon Kelly	5.3	.97	5	23
southwest	SW	Chris Foster	2.7	.8	2	18
southern	SO	May Chin	5.1	.95	4	15
southeast	SE	Derek Johnson	4.0	.7	4	17
eastern	EA	Susan Beal	4.4	.84	5	20
northeast	NE	TJ Nichols	5.1	.94	3	13
north	NO	Val Shultz	4.5	.89	5	9
central	CY	Sheri Watson	5.7	.94	5	13

실례 7-28

```
% nawk '{$9 = $6 * $7; print $9}' datafile
2.94
4.85
1.6
3.8
2.8
4.2
2.82
4.45
4. 7
```

설명

여섯번째 마당($6)과 일곱번째 마당($7)의 곱하기결과는 새로운 마당 $9에 기억되고 인쇄된다. 거기에는 8개의 마당들이 있었는데 지금은 9개 마당이 있다.

실례 7-29

```
% nawk '{$10 = 100; print NF,$9,$0} datafile
10.  northwest   NW   Joel Craig     3.0   98    3   34   100
10.  western     WE   Sharon Kelly   5.3   .97   5   23   100
10.  southwest   SW   Chris Foster   2.7   .8    2   18   100
10.  southern    SO   May Chin       5.1   .95   4   15   100
10.  southeast   SE   DerekJohnson   4.0   .7    4   17   100
```

```
10.  eastern    EA  Susan Beal    4.4  .84  5  20  100
10.  northeast  NE  TJ Nichols    5.1  .94  3  13  100
10.  north      NO  Val Shultz    4.5  .89  5  9   100
10.  central    CT  Sheri Watson  5.7  .94  5  13  100
```

설명

매 레코드에 대하여 열번째 마당($10)은 100으로 대입된다. 이것은 새로운 마당이다. 아홉번째 마당이 존재하지 않으므로 그것은 빈값마당으로 본다. 마당번호가 인쇄되고 그뒤에 NULL마당인 $9의 값, 전체 레코드($0)가 인쇄된다. 열번째 마당값은 100이다.

3. BEGIN패턴

실례 7-30

```
% nawk 'BEGIN{print "- - - - - - - - - - - EMPLOYEES- - - - - - - - - -"}'
- - - - - - - - - - - - - EMPLOYEES- - - - - - - - - - - - -
```

설명

BEGIN패턴뒤에 동작블로크가 놓인다. 동작은 입력파일을 열기전에 문자렬 _____EMPLOYEES_____를 출력한다. 입력파일은 주어 지지 않았으며 awk가 오유통보를 내보내지 않는다는것을 주의하여야 한다.

실례 7-31

```
%nawk 'BEGIN {print"\ t\ t----------ENPLOYESS-------------\ n"\ }\
 {print $0}' datafile
              - - - - - - - - -EMPLOYEES- - - - - - - - - - -

northwest   NW   Joel Craig     3.0   98    3   34
western     WE   Sharon Kelly   5.3   .97   5   23
southwest   SW   Chris Foster   2.7   .8    2   18
southern    SO   May Chin       5.1   .95   4   15
southeast   SE   Derek Johnson  4.0   .7    4   17
eastern     EA   Susan Beal     4.4   .84   5   20
northeast   NE   TJ Nichols     5.1   .94   3   13
north       NO   Val Shultz     4.5   .89   5   9
central     CT   Sheri Watson   5.7   .94   5   13
```

설명

BEGIN동작블로크가 먼저 실행된다. 제목 _____EMPLOYEES_____가 인쇄된다. 두번째 동작블로크는 입력파일에 있는 매 레코드들을 인쇄한다. 행들을 나눌때 거꿀빗선 \ 은 되돌이(CR)를 억제하기 위해 사용되였다. 행들은 반두점 또는 대괄호로 나눌수 있다.

```
% cat datafile2
Joel Craig:northwest:        NW:  3.0  : .98  : 3 : 4
Sharon Kelly:western:        WE:  5.3  : .97  : 5 : 23
Chris Foster:southwest:      SW:  2.7  : .8   : 2 : 18
May Chin:southern:           SO:  5.1  : .95  : 4 : 15
Derek Johnson:southeast:     SE:  4.0  : .7   : 4 : 17
Susan Beal:eastern:          EA:  4.4  : .84  : 5 : 20
TJ Nichols:northeast:        NE:  5.1  : .94  : 3 : 13
Val hultz:north:             NO:  4.5  : .89  : 5 : 9
Sheri Watson:central:        CY:  5.7  : .94  : 5 : 13
```

실례 7-32

% **nawk -BEGIN{ FS=" : ";OFS="\ t"} ;/Charon/{print $1, $2, $8 }'**
datafile2
Sharon Kelly ' western : 28

설명

BEGIN동작블로크는 변수들을 초기화하는데 사용된다. FS변수(마당분리기호)는 옹근두점으로 대입된다. OFS변수(출력마당분리기호)가 타브(\ t)로 대입된다. 레코드가 정규식 sharom으로 시작되면 첫번째, 두번째, 여덟번째 마당($1, $2, $8)이 인쇄된다. 출력의 매 마당은 타브로 구분된다.

4. END패턴

```
% cat datafile
northwest    NW   Joel Craig       3.0   .98   3    4
western      WE   Sharon Kelly     5.3   .97   5   23
southwest    SW   Chris Foster     2.7   .8    2   18
southern     SO   May Chin         5.1   .95   4   15
southeast    SE   Derek Johnson    4.0   .7    4   17
eastern      EA   Susan Beal       4.4   .84   5   20
northeast    NE   TJ Nichols       5.1   .94   3   13
north        NO   Val Shultz       4.5   .89   5    9
central      CY   Sheri Watson     5.7   .94   5   13
```

실례 7-33

% **nawk 'END{print "The total number of records is " NR}' datafile**
The total number of records is 9

설명

awk가 입력파일처리를 끝낸후에 END블로크에 있는 명령문들이 실행된다. 문자렬 The total number of records is가 인쇄되고 그뒤에 마지막 레코드번호 NR의 값이 인쇄된다.

실례 7-34

```
% nawk '/^north/{count++}END{print count}' datafile
3
```

설명

레코드가 정규식 north로 시작되면 사용자정의변수 count가 하나씩 증가한다. awk가 입력파일처리를 끝낸후에 변수 count에 기억된 값이 인쇄된다.

5. BEGIN과 END를 리용한 awk스크립트

실례 7-35

```
 # Second awk script-- awk.sc2 1
1. BEGIN{ FS=":"; OFS="\ t"
      print " NAME\ t\ tDISTRICT\ tQUANTITY"
      print "______________________________\ n"
      }
2.  (print $l"\  t  " $3"\  t \  t" $7)
    (total+=$7) /north/{count++}
3. END{
      print "-------------------------------------------"
      print "The total quantity is " total
      print "The number of northern salespersons is " count
    }
```

(The Output)

4 % **nawk -f awk.sc2 datafile2**

NAME	DISTRICT	QUANTITY
Joel Craig	*NW*	*4*
Sharon Kelly	*WE*	*23*
Chris Foster	*SM*	*18*
May Chin	*SO*	*15*
Derek Johns on	*SE*	*17*
Susan Beal	*EA*	*20*
TJ Nichols	*NE*	*13*
Val Shultz	*NO*	*9*
Sheri Watson	*CT*	*13*

The total quantity is 132
The number of northern salespersons is 3.

설명

1. 우선 BEGIN블로크가 실행된다. 마당분리기호(FS)와 출력마당분리기호(OFS)가 설정된다. 머리부출력이 인쇄된다.
2. awk스크립트의 본체는 datafile2에서부터 들어 오는 입력의 매행에 대해 실행되는 명령문들을 포함한다.
3. END블로크에서 명령문들은 입력파일이 닫긴 다음 즉 awk가 탈퇴하기전에 실행된다.
4. 지령행에서 nawk프로그람이 실행된다. f추가선택뒤에 스크립트이름 awk.scx와 입력파일 datafile2가 놓인다.

```
% cat datafile2
Joel Craig:northwest:          NW: 3.0 : .98 : 3 : 4
Sharon Kelly:western:          WE: 5.3 : .97 : 5 : 23
Chris Foster:southwest:        SW: 2.7  : .8 : 2 : 18
May Chin:southern:             SO: 5.1 : .95 : 4 : 15
Derek Johnson:southeast:       SE: 4.0  : .7 : 4 : 17
Susan Beal:eastern:            EA: 4.4 : .84 : 5 : 20
TJ Nichols:northeast:          NE: 5.1 : .94 : 3 : 13
Val hultz:north:               NO: 4.5 : .89 : 5 : 9
Sheri Watson:central:          CY: 5.7 : .94 : 5 : 13
```

6. printf함수

실례 7-36

```
% nawk '{printf  "$%6.2f\ n", $6 * 100}' datafile
$ 98.00
$ 97.00
$ 80.00
$ 95.00
$ 70.00
$ 84.00
$ 94.00
$ 89. 00
$ 94.00
```

설명

printf함수는 총체적으로 류동소수점수가 6개의 자리수로 오른쪽정돈되게 형식화하는데 여기서 한자리는 소수점, 2개는 소수점아래 두자리수를 위한것이다. 그 수자는 둥그리기되여 인쇄된다.

실례 7-37

```
% nawk  '{printf  "|%-15s|\ n", $4}' datafile
/Craig        /
/Kelly        /
/Foster       /
/Chin         /
/Johnson      /
/Beal         /
/Nichols      /
/Shultz       /
/Watson       /
```

설명

15개 공백문자렬들이 왼쪽정돈되여 인쇄된다. 네번째 마당($4)은 공간을 표시하기 위해 수직막대기안에 놓여 인쇄된다.

7. 방향바꾸기와 파이프

실례 7-38

```
% nawk '/north/{print $1, $3, $4 > "districts"}' datafile
% cat districts
northwest Joel Craig .
northeast TJ Nichols
north Val Shultz
```

설명

레코드에 정규식 north가 있으면 첫번째, 세번째, 네번째 마당이 districts라는 출력파일에 인쇄된다. 파일이 일단 열리면 프로그람이 끝나거나 닫길 때까지 열린 그대로 있다. 파일이름 "districts"는 2중인용부호안에 있어야 한다.

실례 7-39

```
% nawk '/south/{print $1, $2, $3 » "districts"}' datafile
% cat districts
northwest Joel Craig
northeast TJ Nichols:
north Val Shultz
southwest SW Chris
southern SO
southeast SE Derek
```

설명

레코드가 패턴 south를 가지고 있으면 첫번째, 두번째, 세번째 마당($1, $2, $3)

이 출력파일 districts에 추가된다.

8. 파이프열기와 닫기

실례 7-40

```
# awk script using   pipes – awk.sc3
1.   BEGIN
2.        printf " %-22s%s\ n", "NAME", "DISTRICT"
          print --------------------------------------
3.      }
4.   /west/{count++}
5.    {printf "%s %s\ t\ t%-15s\ n". $3, $4, $1| "sort +1" }
6.   END{
7.        close "sort +1"
          printf "The number of sales persons in the western
          printf "region is " count "."}
```

(The Output)

```
% nawk -f awk.sc3 datafile
1.   NAME                                    DISTRICT
2.   --------------------------------------------------------------------
3.   Susan Beal           eastern
     May Chin             southern
     Joel Craig           northwest
     Chris Foster         southwest
     Derek Johnson        southeast
     Sharon Kelly         western
     TJ Nichols           northeast
     Val Shultz           north
     Sheri Watson         central
     The number of sales persons in the western region is 3.
```

설명

1. 특수한 BEGIN패턴뒤에 동작블로크가 놓인다. 이 블로크에 있는 명령문은 awk가 입력파일을 처리하기전에 먼저 실행된다.
2. Printf함수는 문자렬 NAME을 22개의 문자로 왼쪽정돈된 문자렬로 표시하는데 그뒤에 오른쪽정돈된 문자렬 DISTRICT가 놓인다.
3. BEGIN블로크가 끌난다.
4. 현재 awk는 한번에 한행씩 입력파일을 처리한다. 패턴 west를 발견하면 동작블로크는 실행된다. 즉 사용자정의변수 count가 하나씩 증가된다. awk는 처음에 count변수와 만나고 그것이 작성되며 0의 초기값이 주어 진다.

5. print함수는 출력을 형식화하여 파이프에 보낸다. 모든 출력이 수집된후 sort지령에 보낸다.
6. END블로크가 시작된다.
7. 파이프(sort+1)는 그것을 연것과 같은 지령으로 닫겨 져야 한다. 즉 이 실례에서는 “sort+1”이다. 그렇지 않으면 END명령문들은 출력의 나머지와 함께 정렬된다.

UNIX도구에 대한 련습

련습문제 5: nawk련습

```
Mike Harrington:(510)548-1278:250:100:175
Christian Dobbins:(408) 538-2358:155:90:201
Susan Dalsass:(206 654-6279:250:60:50
Archie McNichol:(206)548-1348:250:100:175
Jody Savage :(206) 548-1278:15:188:150
Guy Quigley:(916) 343-6410:250:100:175
Dan Savage:(406) 298-7744:450:300:275
Nancy McNeil:(206) 548-1278:250:80:75
John Goldenrod:(916 348-4278:250:100:175
Chet Main:(510) 538-5258:50:95:135
Tom Savage:(408) 926-3456:250:168:200
Elizabeth Stachelin:(916) 440-1763:175:75:300
```

(CD에서 lab5.data라는 자료기지를 참조하시오.)
우의 자료기지에는 이름과 전화번호, 석달동안의 기부금이 있다.

다음의 보고서를 내보내는 nawk스크립트를 쓰시오.

```
%nawk –f nawk.sc db
 Mike Harrington: (510) 548-1278:250:100 175
Christian Dobbins:(408) 538-2358 155.90:201
Susan Dalsass:(206) 654-6279 250 60:50
Archie McNichol:(206) 548-1348 :250:100-175
Jody Savage:(206)548-1278:15:188:150
Guy Quiglo (916)343-6410 250 100 175
Dan Saverage(206) 298-7744:450:300:275
Nancy McNeil(206) 548-1278:250:100:175
John Goldenrod:(916).348-4278:250:100:175
Chet Main:(510)548-5258:50:95:135
Tom savage:(408)926-3456:250:168:200
Elizabeth Stachelin:(916)440-1763:175:75:300
```

(CD의 lab6.data라는 자료기지를 참조하시오.).

우의 자료기지는 이름, 전화번호, 석달에 걸치는 현금기부금을 포함한다.

1. 다음의 보고서를 제출하는 nawk스크립트를 쓰시오.

%nawk –f nawk.sc db

CAMPAIGN 1998 CONTRIBUTIONS

NAME	PHONE		Jan	Fev	Mar	TotalDonated
Mike Harrington	(510)	548-1278	250.00	100.00	175.00	525.00
Christian Dobbins	(408)	538-2358	155.00	90.00	201.00	446.00
Susan Dalsass	(206)	654-6279	50.00	60.00	50.00	360.00
Archie McNichol	(206)	548-1348	250.00	100.00	175.00	525.00
Jody Savage	(206)	548-1278	15.00	188.00	150.00	353.00
Guy Quigley-	(916)	343-6410	250.00	100.00	175.00	525.00
Dan Savage--	(406)	298-7744	450.00	300.00	275.00	1025.00
Nancy McNeil	(206)	548-1278	250.00	80.00	75.00	405.00
John Goldenrod	(916)	348-4278	250.00	100.00	175.00	525.00
Chet Main	(510)	548-5258	50.00	95.00	135.00	280.00
Tom Savage	(408)	926-3456	250.00	168.00	200.00	618.00
Elizabeth Stacbelin	(406)	440-1763	175.00	75.00	300.00	550.00

SUMMARY

The campaign received a total of $6137.00 for this quarter.
The average donation for the 12 contributors was $511.42.
The highest contribution was $300.00.
The lowest contribution was $15.00.
THANKS Dan

제 6 절. 조건명령문

awk에서 조건명령문은 C언어로부터 나왔다. 이 명령문은 결심채택에서 프로그람의 흐름을 조종하기 위해 사용된다.

1. if명령문

if구조로 시작하는 명령문은 동작명령문이다. 조건패턴으로서 if를 사용한다. 즉 조건동작명령문으로서 if를 명백히 규정하고 그뒤에 괄호안에 표현식이 놓인다. 표현식이 참으로 평가되면(0이 아니거나 빈값이 아닌) 표현식다음에 놓이는 명령문이나 명령블로크가 실행된다. 만일 하나이상의 조건식에 따르는 명령문들이 있으면 그 명령문들은 반두점 혹은 행바꾸기로 구분되며 명령모임은 블로크처럼 실행되도록 대괄호안에 있어야 한다.

형식

```
If (표현식) {
    명령문;명령문;...
}
```

실례 7-41

```
1  % nawk '{if ( $6 > 50 ) print $1 "Too high"}' filename
2  % nawk '{if ($6 > 20 && $6   <= 50){sa£e++; print "OK"}}' filename
```

설명

1. if동작블로크에서 표현식이 검사된다. 여섯번째 마당이 50보다 더 크면 print명령문이 실행된다. 표현식에 뒤따르는 명령문이 단일명령문이므로 대괄호는 필요 없다(filename은 입력파일을 표시한다.).
2. if동작블로크에서 표현식이 검사된다. 여섯번째 마당이 20보다 더 크고 50과 같거나 작으면 표현식뒤에 놓이는 명령문이 블로크로서 실행되며 대괄호안에 들어 가야 한다.

2. if/else명령문

if/else명령문은 두가지 방법을 결정하는데 사용한다. if예약어뒤에 오는 표현식이 참이면 그 표현식과 관련된 명령문블로크가 실행된다.

첫 표현식이 0이거나 거짓으로 평가되면 else예약어뒤에 있는 명령블로크가 실행된다. 다중명령문이 if나 else에 포함되여 있으면 대괄호를 사용하여야 한다.

형식

```
if {상태식} {
     명령문;명령문;...
   }
else {
     명령문;명령문;...
   }
```

실례 7-42

```
1.  % nawk '{i£( $6 > 50) print    $1 " Too high" ;\
              else print "Range is OK")' filename
2.  % nawk "(if ( $6 > 50) { count++; print $3 }  \
              else { x+5; print $2 } }' filename
```

설명

1. 첫 표현식이 참 즉 여섯번째 마당($6)이 50보다 더 크면 print함수는 첫 마당과 Too high를 인쇄한다. 그렇지 않으면 else뒤의 명령문 "Range is OK"가 인쇄된다.

2. 첫 표현식이 참 즉 여섯번째 마당이 50보다 크면 명령문블로크가 실행된다. 그렇지 않으면 else뒤의 명령문블로크가 실행된다. 블로크가 대괄호안에 놓인다는것을 주의하여야 한다.

3. if/else와 else if명령문

if/else와 else if명령문은 여러갈래조건명령문이다. 예약어 if다음에 있는 식이 참이면 그 식과 관련되는 명령블로크가 실행되고 마지막 else와 관련된 마지막 닫는 대괄호뒤에서 다시 조종이 시작된다. 그렇지 않으면 조종은 else if에로 넘어 가고 그 식이 검사된다. 첫 else if조건이 참일 때 그 표현식다음의 명령문이 실행된다. 조건식들중 어느 하나도 참이 아니면 조종은 else명령문에로 넘어 간다. 다른 명령문들이 참이 아니면 else블로크가 실행되므로 else를 기정동작이라고 부른다.

형식

```
if {상태식} {
명령문;명령문;...
}
else if {상태식}{
명령문;명령문;...
}
else if {상태식};...
명령문;명령문;...
}
else{
        명령문
}
```

실례 7-43

(In the Script)

```
1.  {if ( $3 > 89 && $3 < 101 ) Agrade++
2.  else if ( $3 > 79 ) Bgrade++
3.  else if ( $3 > 69 ) Cgrade++
4.  else if ( $3 > 59 ) Dgrade++
5.  else Fgrade++
    }
    END{print "The number of failures is" Fgrade }
```

설명

1. if명령문은 하나의 동작을 나타내며 대괄호안에 있어야 한다. 표현식은 왼쪽으로부터 오른쪽으로 평가된다. 첫 표현식이 거짓이면 전체 표현식은 거짓

이고 첫 표현식이 참이면 론리적합(&&)뒤의 표현식이 평가된다. 참이면 변수 Agrade가 하나 증가된다.
2. if예약어뒤의 첫 표현식이 거짓(0)으로 평가되면 else if표현을 검사한다. 그것이 참이면 표현식다음의 명령문이 실행된다. 즉 세번째 마당($3)이 79보다 크면 Bgrade가 하나 증가된다.
3. 첫 2개의 명령문이 거짓이면 else if표현식이 검사되고 세번째 마당($3)이 69보다 크면 Cgrade가 증가된다.
4. 첫 3개의 명령문이 거짓이면 else if표현식이 검사되고 세번째 마당이 59보다 작으면 Bgrade가 증가한다.
5. 우에서 검사된 표현식이 하나도 참이 아니면 else블로크가 실행된다. 대괄호는 동작블로크를 끝낸다. fgrade가 증가된다.

제 7 절. 순환

순환은 조건이 참이면 검사표현식의 뒤에 놓이는 명령문들을 반복적으로 실행하기 위해 사용된다. 흔히 순환들은 레코드안에서 마당들을 끝까지 반복하며 END블로크에서 배렬의 요소들을 끝까지 순환하기 위해 사용된다. awk는 3개의 순환형명령을 가지고 있다. 즉 while순환, for순환, 특수한 for순환을 가지는데 awk배렬에 대해서 취급할 때 구체적으로 론의하기로 한다.

1. while순환

while순환에서 첫 단계는 변수를 초기값으로 설정하는것이다. 그다음 이 값이 while표현식에서 검사된다. 표현식이 참(령이 아니면)으로 평가되면 순환본체가 시작되고 그 본체안의 명령문들이 실행된다. 순환본체안에 하나이상의 명령문이 있으면 그것들은 대괄호안에 있어야 한다. 순환블로크를 끝내기에 앞서 순환표현식을 조종하는 변수를 갱신하여야 하며 그렇지 않으면 순환은 영원히 계속될수 있다. 다음의 실례에서 변수들은 새 레코드가 처리될 때마다 초기화된다. do/while순환은 표현식이 순환본체가 적어도 한번은 실행될 때까지 검사되지 않는것을 제외하고는 while순환과 류사하다.

실례 7-44

```
% nawk '{I = 1; while (I<=NF ) {print NF; $1 ; i+}} }'filename
```

설명

변수가 i가 1로 초기화된다. i가 레코드에서 마당수(NF)와 작거나 같을 때까지 print명령이 실행되는데 이때 i는 하나씩 증가한다. 변수 i가 NF값보다 클 때까지 표현이 다시 검사된다. 변수 i는 awk가 다음레코드를 처리할 때까지 재초기화되지 않는다.

2. for순환

for순환과 while순환은 for순환이 괄호안에 3개의 표현식 즉 초기화표현식, 시험표현식, 시험표현식안에서 변수를 갱신하기 위한 표현식을 요구하는것을 제외하고 기본적

으로는 갈다. awk에서 for순환의 괄호안의 첫 명령문은 오직 한번의 초기화만을 진행할수 있다(C에서는 반점으로 구별되는 다중초기화가 있을수 있다.).

실례 7-45

```
% nawk '{ for ( i = 1; i <= NF; i++) print NF,$i }' filex
```

설명

변수 i는 1로 초기화되고 레코드에서 그것이 마당의 수(NF)와 작거나 같은가를 보기 위해 검사된다. 만일 그렇다면 print함수는 NF의 값과 $i의 값을 인쇄(i앞의 $는 i번째 마당의 번호이다.)하며 그다음 i는 하나씩 증가된다(흔히 for순환은 배렬의 요소들을 통하여 순환하도록 END동작에서 배렬과 함께 사용된다.). 159페지에서 《배렬》을 참고하기 바란다.

3. 순환조종

break와 continue명령문 break명령문은 어떤 조건이 참이면 순환을 중지시킨다. continue명령문은 어떤 조건이 참이면 뒤에 있는 어떤 명령문을 뛰여 넘도록 순환을 조종하며 다음반복을 시작하기 위해 순환의 시작위치로 조종을 되돌린다.

실례 7-46

```
(In the Script)
1.   {for (x=3; x<= nf;X++)
        if ( $x < 0 ){ print "Bottomed out!"; break)
        # breaks out of for loop
        }

2.   {for ( x = 3; x ^ NF; x++ )
      if ( $x == 0 ) { print "Get next item"; continue}
      # starts next iteration of   the for loop
}
```

설명

1. 마당 $x의 값이 0보다 작으면 break명령문은 조종이 순환본체의 닫는 대괄호뒤에 있는 명령문으로 이동한다. 즉 순환에서 탈퇴한다.
2. continue명령문은 조종이 순환의 꼭대기에서 시작하여 for순환의 세번째 표현인 x++에서 순환이 시작되도록 한다.

제 8 절. 프로그람조종명령문

1. next명령문

next명령문은 입력파일로부터 다음입력행을 받아서 awk스크립트의 제일 꼭대기에서 다시 시작한다.

실례 7-47

```
(In Script)
{ if ($1 ~ /Peter/){next}
else {print}
}
```

설명

첫 마당이 peter를 포함하면 awk는 이 행에로 이행하여 입력파일로부터 다음행을 취한다. 스크립트는 처음부터 실행을 다시 시작한다.

2. exit명령문

exit명령문은 awk프로그람을 끝낸다. 이것은 레코드처리를 중지하지만 END명령문에로 이행하지는 않는다. exit명령문이 인수(exit 1)로서 0~255사이의 값이 주어 지면 이 값은 입력의 성공인가 실패인가를 가리키기 위해 지령행에 출력될수 있다.

실례 7-48

```
(In Script)
{exit (1)
(The Command Line)
% echo $status        (csh)
1
$ echo $7      (sh/ksh)
1
```

설명

exit상태 0은 성공을 의미하고 0이 아닌 값은 실패를 의미한다(UNIX의 규칙이다.). 프로그람에 탈퇴상태를 제공하는것은 프로그람작성자에게 달려 있다. 이 프로그람에서 돌려 지는 탈퇴값은 1이다.

제 9 절. 배렬

awk에서 배렬은 보조스크립트가 수자 혹은 문자일수 있기때문에 련상배렬이라고 한다. 보조스크립트는 흔히 열쇠라고 부르며 대응하는 배렬요소에 대입되는 값과 관련된다. 열쇠와 값들은 하쉬(hash)알고리듬의 질문에서 건값에 적응되는 표에 내부적으로 기억된다. 하쉬법에 사용되는 방식때문에 배렬요소들은 순차적인 순서로 기억되지 않으며 배렬의 내용이 표시될 때 요구하는 순서로 되지 않을수 있다.

변수와 같이 배렬은 그것을 사용하여 작성되며 awk배렬은 그것이 수자를 기억하는가 문자렬를 기억하는가를 지정할수 있다. 배렬요소들은 문맥에 따라서 수값 0과 빈값문자렬으로 초기화된다. awk배렬의 크기는 선언하지 않아도 된다. awk배렬들은 레

코드들로부터 정보를 수집하고 합계를 내며 단어를 계수하고 패턴이 발생한 수의 추적 등에 사용된다.

1. 련상배렬에 대한 보조스크립트

변수를 배렬첨수로 리용

실례 7-49

```
(The Input File)
% cat employees
Tom Jones          4424     5/12/66     543354
Mary Adams         5346     11/4/63     28765
Sally Chang        1654     7/22/54     650000
Billy Black        1683     9/23/44     336500

 (The Command Line)
1.  % nawk '{name(x*+]«$2);END{for(i=0; i<NR; i++)\
    print i, nir<i[ i ] ]' employees
    0 Jones
    1 Adams

2   Chang
3.  Black
2.  % nawk '{id[NR]=$3}»BHD{for(x = 1; x <= MR; x++)\
    print id[x]}' employees
    4424
    5346
    1654
    1683
```

설명

1. 배렬 name에서 보조스크립트는 사용자정의변수 x이다. ++는 수자내용을 지적한다. awk는 x를 0으로 초기화하고 사용후에(post증가연산자) x를 하나씩 증가시킨다. 두번째 마당값은 name배렬의 매 요소로 대입된다. END블로크에서 for순환은 거기에 기억된 값들을 인쇄하고 보조스크립트에서 시작하여 배렬의 끝까지 순환하는데 사용된다. 보조스크립트가 바로 하나의 열쇠이기때문에 0에서 시작하지 않아도 된다. 그것은 임의의 값 즉 문자렬이나 수자로 시작할수도 있다.
2. awk변수 NR는 현재레코드번호를 가지고 있다. NR를 보조스크립트로 사용하여 세번째 마당값을 매 레코드에 대하여 배렬의 매 요소에 대입한다. 마지막에 for순환은 레코드에 기억된 값들을 출력하면서 배렬을 끝까지 순환한다.

특수한 for순환 특수한 for순환은 for순환이 실용성이 없는 경우에 련상배렬을 끝까지 읽기 위해 사용된다. 즉 문자렬이 보조스크립트처럼 사용되거나 혹은 보조스크립트가 련속적인 번호가 아닐 때 련상배렬을 사용한다. 특수한 for순환은 그와 련관된 값에로의 열쇠로서 보조스크립트를 사용한다.

형식

```
{for(item in arrayname) {
        print arrayname[item]
        }
}
```

실례 7-50

```
(The Input File)
 % cat db
Tom Jones
Mary Adams
Sally Chang
Billy Black
Tom Savage
Tom Chang
Reggie Steel
Tommy Tucker

(The Command Line, for Loop.).
1.  % nawk '/' 'Tom/{name [NR] =$l};\
    END{for( i = 1; i <= NR; i++ .).print name[i]}' db
    Tom
    Tom
    Tom
    Tommy

    (The Command Line, Special for Loop.).

2.  % nawk '/^Tom/{name[NR]=$1};\
    END{ for(I in name.). { print name[I]}}' db
    Tom
    Tommy
    Tom
    Tom
```

설명

1. 정규식 Tom이 입력행에서 일치되면 name배렬은 하나의 값이 대응된다. 사용되는 보조스크립트가 NR(현재레코드번호)이므로 배렬에서 보조스크립트들은 크기순서로 놓이지 않는다. 그러므로 전통적인 for순환으로 배렬을 인쇄할 때 배렬요소가 값을 가지지 않는 경우 인쇄되지 않는것에는 인쇄되는 빈 값이 있다.

2. 특수한 for순환은 그 값과 관련된 보조스크립트가 있는 값만을 인쇄하면서 배렬을 끝까지 반복한다. 출력순서는 련상배렬이 하쉬법으로 기억되여 있으므로 무질서하다.

배렬보조스크립트로서 문자렬을 사용 보조스크립트는 하나의 문자렬이나 자모문자로만 되여 있는 문자렬이 있는 변수로 이루어 질수 있다. 문자렬이 문자로만 되여 있으면 2중인용부호안에 있어야 한다.

실례 7-51

(The Input File.).

```
% cat datafile3
tom
mary sean
torn
 mary
 mary
bob
 mary
alex
```

(The Script.).

```
# awk.sc script
1.  /torn/ { count "tom"}++ }
2.  /mary/ { count["mary"]++ }
3.  END{print "There are " count["tom"] " Toms in the fil
" count["mary"]" Marys in the file."}
```

(The Command Line.).

```
% nawk -f awk.sc data£ile3
There are 2 Toms In the file and 4 Marys in the file.
```

설명

1. count라고 하는 배렬은 2개의 요소 즉 count["tom"]과 count["mary"]로 구성된다. 배렬요소의 매 초기값은 0이다. Tom이 일치될 때마다 배렬값은 하나씩 증가된다.
2. 동일한 절차가 count["mary"]에 적용된다. 주의: Tom이 행에서 여러번 나타나도 매행에 대하여 한개의 Tom만이 기록된다.
3. END패턴은 매 배렬요소에 기억된 값을 출력한다.

배렬보조스크립트로서 마당값사용 표현식은 배렬에서 보조스크립트로 사용될수 있다. 따라서 마당을 리용할수 있다. 실례 7-52의 프로그람은 두번째 마당에 나타나는 모든 이름들의 출현빈도수를 계수하고 for순환의 새로운 형식을 지정한다.

실례 7-52

(The Input File.).

% cat datafile4

```
4234    Tom     43
4567    Arch    45
2008    Eliza   65
4571    Tom     22
3298    Eliza   21
4622    Tom     53
2345    Mary    24
```

(The Command Line.).

```
% nawk '{countt$2}++}END{for(naine in count)print name,count[name] }'\
    datafile4
Tom 3
Arch 1
Eliza 2
Mary 1
```

설명

awk명령문은 우선 count배렬의 첨수로서 두번째 마당을 사용한다. 첨수는 두번째 마당이 변할 때 변하므로 count배렬에서 첫 첨수는 Tom이고 count["Tom"]에 기억된 값은 1이다. 다음에 count["arch"]는 1, count["Eliza"]는 1, count["Mary"]는 1로 설정된다. awk가 두번째 마당에서 Tom의 다음출현을 발견하면 count["Tom"]이 증가되여 현재값 2를 가진다. Arch, Eliza, Mary가 나타날 때마다 같은 현상이 일어 난다.

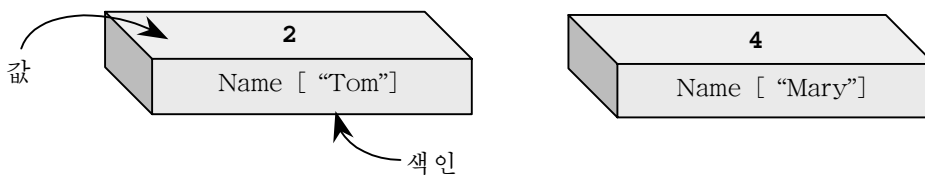



그림 7-1. 배렬의 보조스크립트로서 문자렬사용(실례 7-51)

· for (index_value in array) statement

앞의 실례의 END블로크에 있는 for순환은 다음과 같이 동작한다. 즉 변수 name은 count배렬의 첨수값으로 설정된다. for순환의 매 반복후에 print동작이 수행되여 우선 첨수값, 그다음에 그 요소에 기억된 값을 인쇄한다(출력순서는 담보되지 않는다.).

실례 7-53

(The Input File.).

```
% cat    datafile4
4234        Tom         43
4567        Arch        45
2008        Eliza       65
4571        Tom         22
3298        Eliza       21
4622        Tom         53
2345        Mary        24
```

(The Command Line.).

```
% nawk '{dupt$2)++;   if fdBptSa] > 1){name[$21++ }}\
END{print   "The duplicates Were"\
For (i in name.). {print I,name[I]}}'   datafile4
```

(The Output.).

```
Tom 2
Eliza 2
```

설명

dup배렬에 대한 보조스크립트는 두번째 마당의 값 즉 사람의 이름이다. 거기에 기억되는 값은 처음에 0이며 그것은 새로운 레코드가 처리될 때마다 하나씩 증가된다. 이름이 2개이면 그 보조스크립트에 기억된 값은 2로 된다. dup배렬의 값이 1보다 크면 name이라고 하는 새 배렬도 보조스크립트로 두번째 마당을 사용하며 1보다 더 큰 이름번호를 유지한다.

배렬과 split함수 awk의 내부변수는 문자렬을 단어로 나누어 배렬에 기억한다. 사용자는 마당분리기호를 정의하거나 FS에 기억된 값을 사용할수 있다.

형식

```
split(string, array, field separator)
split (string, array)
```

실례 7-54

(The Command Line.).

```
% nawk BEGIN{ split( "3/15/2001", date, "/".).;   \
print "The month is " date[l] "and the year is "date[3]"}  \
filename
```

(The Output.).

```
The month is 3 and the year is 2001.
```

설명

문자렬 3/15/2001은 마당분리기호로서 거꿀빗선을 사용하여 배렬 date에 기억된다. 현재 date[1]은 3, data[2]는 15, data[3]은 2001을 기억하고 있다. 마당분리기호는 세번째 인수에 지정되며 지정되지 않으면 FS의 값이 분리기호로 사용된다.

delete함수 delete함수는 배렬요소를 삭제한다.

실례 7-55

```
% nawk '{line[x++]=$2}END{for(x in line.). delete(line[x])}'\
filename
```

설명

배렬 line에 대입되는 값은 두번째 마당값이다. 모든 레코드들이 처리된후 특수한 for순환은 배렬의 매 요소를 반복하고 delete함수는 되돌이에서 매 요소를 삭제한다.

다차원배렬(nawk) awk가 공식적으로는 다차원배렬을 지원하지 않지만 다차원배렬을 리용할수 있는 한가지 방식을 제공한다. 이것은 첨수들을 특수한 내부변수 SUBSEP의 값에 의해 분리되는 한개의 문자렬로 련결하여 진행한다. SUBSEP변수에는 인쇄할수 없는 문자인 "\ 034"가 있는데 이것은 잘 사용하지 않기때문에 첨수문자로서는 거의 리용되지 않는다. 표현행렬 [2, 8]은 실지로 배렬행렬 [2 SUBSEP 8]로서 matrix["2\0348"]과 같다. 첨수는 련상배렬에서 유일한 문자렬이다.

실례 7-56

```
(The   Input File.).
1 2 3 4 5
2 3 4 5 6
6 7 8 9 10

(The Script.).
1.   {nf=NF
2.   for (x = 1; x<= NF; x++){
3.   matrix[NR,x] = $x
             }
     }
4.   END { for   (x=1; x<= NR; x++) {
                    for     (y = 1; y <= nf ;y++)
                          printf " %d", matrix[x,y]
printf "\ n"
           }
   }

(The Output.).
1 2 3 4 5
```

```
2 3 4 5 6
6 7 8 9 10
```

설명

1. 변수 nf는 NF 즉 마당번호의 값으로 대입된다(이 프로그람은 한 레코드당 마당의 수를 5개로 고정한다.).
2. for순환이 진행되여 변수 x에 행의 매 마당번호를 기억시킨다.
3. Matrix배렬은 2차원배렬이다. 2개의 첨수들인 NR(현재레코드의 번호)와 x는 매 마당값으로 대입된다.
4. END블로크에서 2개의 for순환은 거기에 기억된 값을 출력하면서 matrix배렬을 반복하기 위해 사용된다. 이 실례는 다차원배렬을 모의할수 있다는것을 확증하는데 불과하다.

2. 지령인수처리(nawk)

NRGV 지령행인수들은 ARGV라는 내부배렬과 함께 nawk(새로운 awk본)에서 리용할수 있다. 이 인수들은 지령 nawk에 넘겨 지는 임의의 추가선택들을 포함하지 않는다. ARGV배렬의 첨수는 0에서부터 시작한다(이것은 오직 nawk에서만 가능하다.).

ARGC ARGC는 지령행인수들의 번호를 기억하는 내부변수이다.

실례 7-57

```
(The Script.).
# This script is called argvs
BEGIN{
for ( i=0; i < ARGC; i++ .).{
printf("argv[%d] is %s소n", i, ARGV[i])
}
printf("The number of arguments, ARGC=%d\ n", ARGC.).
}
(The Output.).
% nawk -f argvs datafile
argv[0] is nawk
arg~v[l] is datafile
The number of arguments, ARGC=2
```

설명

For순환에서 i는 0으로 설정되고 지령행인수번호(ARGC)보다 작은가를 검사하며 printh함수는 차례로 나타나는 매 인수들을 표시한다. 모든 인수들이 처리된 후 마지막 printf명령문은 인수번호(ARGC)를 출력한다. 실례는 awk가 지령행추가선택들을 인수로 하지 않는다는것을 보여 준다.

실례 7-58

(The Command Line.).
% **nawk -f argvs datafile "Peter Pan" 12**
argv[0] is nawk
argv[1] is datafile
argv[2] is Peter Pan
argv [3] is 12
The number of arguments, ARGC=4

설명

마지막실례에서와 같이 매 인수는 인쇄된다. nawk지령은 첫번째 인수로 되고 –f추가선택스크립트이름 argvs는 제외된다.

실례 7-59

(The Datafile.).
% **cat datafile5**
Tom Jones:123:03/14/56
Peter Pan:456:06/22/58
Joe Blow:145:12/12/78
Santa Ana:234:02/03/66
Ariel Jones:987:11/12/66

(The Script.).

% **cat arging.ac**
 # This script is called arging.sc
1. ***BEGIN{FS=":"; nainc=ARGV[2]***
2. **print "ARGV[2] is "ARGV[2]**
}
$1 ***- name { print $0 }***

(The Command Line.).
% **nawk -f arging.sc data£ile5 "Peter Pan"**
ARGV[2] is Peter Pan
Peter Pan:456:06/22/58
nawk: can't open Peter Pan
input record number 5, file Peter Pan
source line number 2

설명

1. BEGIN블로크에서 변수 name에는 ARVG[2]의 값 Peter pan이 대입된다. peter pan은 인쇄되지만 awk는 datafiles을 처리하고 닫은후 입력파일로 Peter Pan을 열려고 시도한다. awk는 인수를 입력파일로 한다.

실례 7-60

(The Script.).

```
% cat arging2.sc
BEGIN{FS=" : " ; name=ARGV[2]
print "ARGV[2] is - ARGV[2]
delete ARGV[2]
}
$1 ~ name { print $0 .).
```

(The Command Line.).

```
% nawk -f arging2.sc datafile "Peter Pan"
ARGV[2] is Peter Pan
Peter Pan: 456: 06/22/58
```

설명

awk는 ARGV배렬요소를 입력파일로 한다. 즉 인수를 사용한후 그것을 왼쪽으로 옮기고 ARGV배렬이 빌 때까지 다음의 인수들을 처리한다. 만일 인수가 사용된후 즉시에 삭제되면 다음의 입력파일로 처리될수 없다.

제 10 절. awk내부함수

1. 문자렬함수

sub와 gsub함수 sub함수는 정규식을 레코드에 있는 가장 크고 제일 왼쪽에 있는 부분문자렬과 비교한 다음 그 부분문자렬을 대치문자렬로 치환한다. 목적문자렬이 지정되면 정규식은 목적문자렬에 있는 가장 크고 제일 왼쪽에 있는 부분문자렬과 일치되여 그 부분문자렬은 대치문자렬과 치환된다. 목적문자렬이 지정되지 않으면 전체 레코드가 사용된다.

형식

sub (regular expression, substitution string.).

sub (regular expression, substitution **string,** target string.).

실례 7-61

1. **% nawk '{sub(/Mac/, "Macintosh".).; print}' filename**
2. **% nawk '{sub(/Mac/, "Macintosh", $1.).; print}' filename**

설명

1. 처음에 정규식 Mac가 레코드($0)에서 일치되면 그 정규식은 문자렬 "MacIntosh"로 바뀐다. 치환은 행에서 첫 일치때에만 일어 난다(다중사건에 대해서는 gsub를 참고하기 바란다.).
2. 정규식 Mac가 레코드의 첫 마당에서 처음으로 일치될 때 문자렬 MacIntosh로 치환된다. 치환은 목적문자렬에 대한 행에서 첫번째로 일치될 때에만 진행된다. gsub함수는 정규식을 문자렬로 모두 교체한다. 즉 정규식이 매 레코드($0)에서 일치되는 매 사건마다 교체한다.

형식

Gsub(regular expression, substitution string.).
Gsub(regular expression, substitution string,target string.).

실례 7-62

1. **% nawk '{ gsub(/CA/, "California".).; print }' datafile**
2. **% nawk '{ g«ub(/ [Tt] om/, "ThoBias", $1 .).; print }' filename**

설명

1. 레코드($0)에서 정규식 CA가 있는 곳마다에서 정규식 CA는 문자렬 California로 교체된다.
2. 정규식 Tom이나 tom이 첫 마당에서 발견되는 곳마다에서 그 표현은 문자렬 Thomas로 교체된다.

Index함수 index함수는 부분문자렬을 문자렬에서 발견되는 첫 위치로 되돌린다. 편차는 위치 1에서 시작한다.

형식

index (string, substring)

실례 7-63

% nawk '{print index("hollow", "low".). }' filename
4

설명

되돌려 지는 수는 hollow에서 부분문자렬 low가 문자렬의 시작으로부터 발견되는 위치인데 편차는 1에서 시작한다.

Length함수 Length함수는 문자렬에서 문자의 수를 되돌려 준다. 인수없이 length함수는 레코드에서 문자의 수를 되돌려 준다.

형식

```
length (string)
length
```

실례 7-64

% nawk '{ print length("hellow".). }' filename

설명

length함수는 문자렬 hellow에서 문자의 수를 되돌려 준다.

substr함수 substr함수는 첫 위치가 1인 위치에서 시작하는 문자렬의 부분문자렬을 되돌려 준다. 부분문자렬의 길이가 주어 지면 문자렬의 그 부분이 되돌려 진다. 등록된 길이가 실제문자렬을 초과하면 그 문자렬은 되돌려 진다.

형식

```
substr(string,string position.).
substr(string,starting position,length of string.).
```

실례 7-65

% nawk ' { print substr("Santa Claus", 7,6 .).}' filename
Claus

설명

문자렬 Santa Class에서 문자렬의 길이가 6인 위치 7에서 시작하는 부분문자렬을 인쇄한다.

match함수 match함수는 정규식의 문자렬에서 발견되는 첨수를 되돌리며 발견되지 않으면 0을 되돌려 준다. match함수는 내부변수 RSTART를 문자렬에 있는 부분문자렬의 시작위치로 설정하고 RLENGTH를 부분문자렬의 끝까지의 문자들의 개수로 설정한다. 이 변수들은 패턴을 추출하기 위해 substr함수와 함께 사용된다(오직 nawk에서만 가능하다.).

형식

```
Match(string,regular expression)
```

실례 7-66

% nawk 'END{start=match("Good ole USA", /[A-Z]+$/); print start}'\ filename
10

설명

정규식 /[A-Z]+$/는 문자렬끝에서 련속적인 대문자들을 탐색하게 한다. 부분문자렬 USA는 문자렬 Good ole USA의 10번째 문자에서 시작된다. 문자렬이 일치하지 않으면 0을 되돌린다.

실례 7-67

1. % **nawk 'END{start=match("Good ole USA", /[A-Z]+$);**

   print RSTART, RLENGLISH}' filename

   10 3

2. % **nawk 'BEGIN{ line="Good ole USA"}; **

   END{ match(line, /[A-Z]+$/);\

   print substr(line, RSTART, RENGLISTH) } ' filename

   USA

설명

1. RSTART변수는 match함수에 의해 일치되는 정규식의 시작위치로 설정된다. RSTART변수는 부분문자렬의 길이로 설정된다.
2. substr함수는 변수 line에서 부분문자렬을 탐색하기 위해 사용되며 시작위치와 부분문자렬의 길이로서 RSTART와 RLENGTH값(match함수에 의해 설정된다.)을 사용한다.

split함수 split함수는 어떠한 마당분리기호가 세번째 파라메터로 지정되든지 관계없이 그것을 리용하여 문자렬을 배렬로 나눈다. 세번째 파라메터가 제공되지 않으면 awk는 FS의 현재값을 사용한다.

형식

split (string, array, field separator)
split (string, array)

실례 7-68

% **awk 'BEGIN{split("12/25/2001", "/");print date[2]}' filename**
25

설명

split함수는 분리기호로서 빗선문자 /를 사용하여 문자렬 12/25/2001을 date라고 부르는 배렬로 나눈다. 배렬보조스크립트는 1로부터 시작된다. date배렬의 두번째 요소가 인쇄된다.

sprintf함수 sprintf함수는 지적된 형식으로 표현을 되돌려 준다. 이것은 사용자들이 printf함수의 형식지정(specification)을 응용할수 있도록 한다.

형식

```
Variable=sprintf(:string with fomart specifiers", expr1,expr2, \
… , expr2)
```

실례 7-69

```
% awk '{line = sprintf ( "%-15s %6.2f", $1, $3 .).;\
print line }' filename
```

설명

첫번째와 두번째 마당들은 printf지정에 따라 형식화된다(왼쪽정돈된 15개의 공백문자렬과 오른쪽정돈된 6자리의 류동소수점수). 결과는 사용자정의변수 line에 대입된다. 149페지에서 《printf함수》를 참고하기 바란다.

제 11 절. 내부산수함수

표 7-3은 x와 y가 임의의 표현인 내부산수함수를 보여 주었다.

표 7-3. 산수식함수

이 름	되돌이값
atan2(x,y)	범위에서 y/x의 아크탕겐스값이다.
cos(x)	x(라디안)의 코시누스값이다.
exp(x)	e의 x제곱함수이다.
int(x)	x>0일 때 x의 옹근수부이다.
log(x)	x의 자연로그함수이다.
Rand()	0<r<1인 자연수 r이다.
sin(x)	x의 시누스값이다.

(표계속)

이 름	되돌이값
sqrt(x)	x의 뿌리이다.
srand(x)	x는 rand()에 대한 새로운 seed이다.ㄱ

ㄱ. Alfred V.Aho, Brian W.Kernighan, Peter J.Weinberger, AWK 프로그람작성언어 (Boston: Addson-Wesley, 1988).©1988 Bell Telephone Laboratories, Inc.Reprinted by permission of Pearson Education, Inc.

1. 옹근수함수

int함수는 소수점의 오른쪽에 있는 임의의 수자를 잘라서 옹근수를 만든다. 자리올림을 하지 않는다.

실례 7-70

```
1.  % awk 'END{print 31/3}' filename
     10.3333
2.  % awk 'END{print int(31/3)}' filename
     10
```

설명

1. END블로크에서 나누기결과는 류동소수점수이다.
2. END블로크에서 int함수는 나누기결과가 소수점에서 잘리우게 한다. 옹근수는 표시된다.

2. 우연수발생기

rand함수 rand함수는 0보다 크거나 같고 1보다 작은 가상우연수의 류동소수점수를 발생시킨다.

실례 7-71

```
% nawk '{print rand()}' filename
0.51387:
0.175726
0.308534

% nawk  '{print rand()}' filename
0.513871
0.175726
0.308634
```

설명

프로그람이 실행될 때마다 같은 수들의 모임이 인쇄된다. srand함수는 새로운

시작값을 가지고 rand함수가 동작하게 하는데 사용할수 있다. 그렇지 않으면 이 실례에서와 같이 rand가 호출될 때마다 같은 수렬이 반복된다.

srand함수 인수가 없는 srand함수는 시간을 리용하여 rand함수에 대한 시작값을 발생시킨다. srand(x)는 시작값으로서 x를 쓴다. 일반적으로 x는 프로그람실행기간 변해야 한다.

실례 7-72

% nawk 'BEGIN{srand(.).};{print rand()}' filename
0.508744
0.639485
0.657277

% nawk ‘BEGIN{srand(.).};{print rand()}’ filename
0.133518
0.324747
0.691794

설명

srand함수는 rand에 새로운 시작값을 설정한다. 시작점은 시간이다. rand가 호출될 때마다 새로운 수렬들이 인쇄된다.

실례 7-73

% nawk ‘BEGIN{srand(.).};{print 1 + int(rand()’ 25.)..}.' filename
6
000000000000000000000000000000
24
14

설명

srand함수는 rand에 새로운 값을 설정한다. 시작점은 시간이다. rand함수는 0으로부터 255사이의 우연수를 선택하고 옹근수로 계산한다.

제 12 절. 사용자정의함수

사용자정의함수는 패턴동작규칙이 적용되는 스크립트의 임의의 위치에 놓일수 있다.

형식

```
function name ( parameter, parameter, parameter, ...) {
        statements
        return expression
```

```
(The return statement and expression are optional.).
}
```

변수들은 값에 의해 넘겨 지며 사용되는 함수에 국한된다. 변수들의 복사만을 사용한다. 배렬은 주소나 참고에 의해 넘겨 지므로 배렬요소들은 함수안에서 직접 변화될수 있다. 파라메터목록에서 넘겨 지지 않은 함수안에서 사용되는 임의의 변수를 전역변수로 본다. 즉 이 변수는 전체 awk프로그람에서 리용할수 있으며 함수에서 그 값이 변화되면 프로그람전체에서도 변화된다. 함수안에서 국부변수를 제공하는 유일한 방법은 파라메터목록에 그것들을 추가하는것이다. 이러한 파라메터들은 보통 목록의 끝에 놓인다. 만일 함수호출에서 제공되는 형식적파라메터가 없으면 파라메터는 처음에 빈 값으로 설정된다. 되돌이명령문은 조종과 가능한 값을 호출자에게 되돌려 준다.

실례 7-74

```
(The Command Line Display of grades File before Sort.).
% cat grades
44 55 66 22 77 99
100 22 77 99 33 66
55 66 100 99 88 45

(The Script.).
% cat sorfr.sc
# Script is called sorter
# It sorts numbers in ascending order
1.  function sort ( scores,num_elements, temp,   I, j .). {
        # temp, I,and   j will be local and private,
        # with an initial value of null.
2.  for   ( i= 2; I <= num+elements; ++I .). {
          3.         for ( j=I; scores [j-1] > scores[j]; --j .). }
                        temp = scores[j]
                        scores[j] =scores[j-1]
                        scores[j-1] = temp
                  }
          4.   }
5.  }
6.  {for ( i = 1 ; I <= NF ;    I++)
          grades[I]=$I
    7.  sort(grades, NF.).            # Two arguments are passed
    8.  for ( j = 1; j <= NF; ++j .).
            print( "%d", grades[j] .).
            print("\  n".).
        }
```

(After the Sort.).
% **nawk –f sorter.sc grades**
22 44 55 66 77 99
22 33 66 77 99 100
45 55 66 88 99 100

설명

1. sort라는 함수를 정의한다. 함수는 스크립트의 임의의 위치에서 정의될수 있다. 모든 변수들은 파라메터로 넘겨 지는것들을 제외하고 전역변수이다. 만일 전역변수들이 함수에서 변화되면 전체 awk스크립트에서 변화된다. 배열은 참고에 의해 넘겨 진다. 5개의 형식인수들은 괄호 ()안에 놓인다. 배열 score는 참고에 의해 넘겨 져 배열의 임의의 요소들이 함수내에서 변경되면 초기배렬이 변경된다. 변수 num_element는 국부변수로서 초기변수의 복사이다. 변수 temp, i, j는 함수에서 국부변수이다.
2. 외부for순환은 적어도 비교할수 있는 2개의 수가 있는 한 수배렬을 반복한다.
3. 내부for순환은 현재의 수와 이전의 수 scores[j-1]을 비교한다. 만일 이전의 배렬요소가 현재의 수보다 크면 temp는 현재배렬요소의 값을 대입 받고 현재배렬요소에는 이전의 요소의 값이 대입된다.
4. 밖의 순환블로크를 끝낸다.
5. 이것은 함수정의의 끝이다.
6. 스크립트의 첫 동작블로크는 여기서 시작한다. for순환은 수들의 배렬을 작성하여 현재레코드의 매 마당을 반복한다.
7. sort함수가 호출되여 현재의 레코드로부터 수배렬과 마당번호를 넘긴다.
8. sort함수가 완료되면 프로그람조종은 여기에서 시작된다. for순환은 정렬된 배렬의 요소들을 인쇄한다.

제 13 절. 개괄

% cat datafile

northwest	NW	Joel Craig	3.0	.98	3	4
western	WE	Sharon Kelly	5.3	.97	5	23
southwest	SW	Chris Foster	2.7	.8	2	18
southern	SO	May Chin	5.1	.95	4	15
southeast	SE	Derek Johnson	4.0	.7	4	17
eastern	EA	Susan Beal	4.4	.84	5	20
northeast	NE	TJ Nichols	5.1	.94	3	13
north	NO	Val Shultz	4.5	.89	5	9
central	CY	Sheri Watson	5.7	.94	5	13

실례 7-75

```
% nawk '{if ( $8 > 15 ).< print $3 “ has a high rating”}소
    else print $3 “——NOT A COMPETITOR---”}’ datafile
Joel --- NOT A COMPETITOR---
Sharon has a high rating
Chris has a high rating
May ---NOT A COMPETITOR---
Dreck has a high rating
Susan has a high rating
TJ---NOT A COMPRTIROR---
Val---NOT A COMPRTIROR---
Sheri---NOT A
```

설명

if명령문은 동작명령문이다. 표현뒤에 하나이상의 명령문이 있으면 그것들은 대괄호안에 있어야 한다(이 실례에서는 표현뒤에 명령문이 하나이므로 괄호를 요구하지 않는다.). 표현의 내용은 다음과 같다. 여덟번째 마당이 15보다 크면 세번째 마당과 문자렬 has a high rating을 인쇄하고 15보다 작으면 세번째 마당과 ---NOT A COMETITOR---를 인쇄한다.

실례 7-76

```
% nawk '{I=1;while(I<=NF && NR < 2)   {print $i; i++}}' datafile
northwest
NW
Joel
Craig
3.0
.98
3
4
```

설명

사용자정의변수 i는 1로 대입된다. While순환이 시작되고 표현식을 검사한다. 표현식이 참으로 평가되면 print명령문이 실행되고 i번째 마당의 값이 출력된 다음 i의 값이 하나 증가하고 순환이 다시 시작된다. 순환표현식은 i의 값이 NF보다 더 크고 NR의 값이 둘이상일 때 거짓으로 된다. 변수 i는 다음레코드가 입력될 때까지 재초기화되지 않는다.

실례 7-77

```
% nawk `{ for( i=3 ; i <= NF && NR ==3 ; i++) { print $i }}` datafile
Chris
Foster
2.7
.8
2
18
```

설명

이것은 while순환과 기능적으로는 류사하다. 초기화, 검사순환조종명령문들은 모두 하나의 표현식에 놓인다. i(i=3)의 값은 현재레코드에 대해서 한번 초기화된다. 다음에 표현식이 검사된다. i가 NF보다 작거나 같고 NR가 3과 같으면 print 블로크가 실행된다. i번째 마당값이 인쇄된후 조종은 순환표현식에로 되돌려 진다. i의 값은 증가되고 시험은 반복된다.

```
% cat datafile
northwest    NW    Joel Craig       3.0   .98   3    4
western      WE    Sharon Kelly     5.3   .97   5   23
southwest    SW    Chris Foster     2.7   .8    2   18
southern     SO    May Chin         5.1   .95   4   15
southeast    SE    Derek Johnson    4.0   .7    4   17
eastern      EA    Susan Beal       4.4   .84   5   20
northeast    NE    TJ Nichols       5.1   .94   3   13
north        NO    Val Shultz       4.5   .89   5    9
central      CY    Sheri Watson     5.7   .94   5   13
```

실례 7-78

```
(The Command Line.).
% cat nawk.sc4
# Awk scripts illustrating arrays
BEGIN{OFS="\ t"}
{ list[NR] = $1}        # The array is called list. The index in the
                        # number of the current record. The value of the
                        # first field is assigned to the array element.
END { for( n = 1; n <= NR; n++)   {
        print list[n]}   # for loop is used to loop
```

```
                        # through the array.
(The Command Line.).
   % nawk –f nawk.sc4 datafile
   northwest
   western
   southwest
   southeast
   eastern
   north
   central
```

설명

배열 list는 첨수값으로서 NR를 사용한다. 입력행이 처리될 때마다 첫번째 마당은 list배열에 대입된다. END블로크에서 for순환은 배렬의 매 요소를 반복한다.

실례 7-79

```
(The Command Line.).
% cat nawk.sc5
#Awk script with special for loop
/north/{name[count++]=$3}
END{ print "The number living in a northern district: " count
     print "Their names are: "
     for ( i in name .).          # Special nawk for loop is used to
         print name[i]          # iterate through the array.
}
% nawk -f   nawf.sc5 datafile
The number living in a northern district: 3
Their names are:
Joel
TJ
Val
```

설명

정규식 north가 행에 나타날 때마다 name배렬에 세번째 마당값이 대입된다. 첨수 count는 새 레코드가 처리될 때마다 증가된다. 따라서 배렬에 또 다른 요소들이 산생된다. END블로크에서 특수한 for순환은 배렬을 반복하기 위해 사용된다.

```
% cat datafile
northwest    NW   Joel Craig      3.0   .98   3    4
western      WE   Sharon Kelly    5.3   .97   5   23
southwest    SW   Chris Foster    2.7   .8    2   18
southern     SO   May Chin        5.1   .95   4   15
southeast    SE   Derek Johnson   4.0   .7    4   17
eastern      EA   Susan Beal      4.4   .84   5   20
northeast    NE   TJ Nichols      5.1   .94   3   13
north        NO   Val Shultz      4.5   .89   5    9
central      CY   Sheri Watson    5.7   .94   5   13
```

실례 7-80

```
(The Command Line.).
% cat nawk.sc6
        # Awk and the special for loop
        {region[$1]++}   # The index Is the first field of each record
END{for(item in region.). {
            print region [item] , item
             }
        }

        % nawk –f   `nawk.sc6 datafile
        1 central
1 northwest
1 western
1 southeast
1 north                .
1 southern
1 northeast
1 southwest
1 eastern

% nawk -f nawk.sc6 datafile3
4 Mary
2 Tom
1 Alax
1 Bob   .
1 Sean
```

설명

region배렬은 첨수로서 첫 마당을 사용한다. 기억된 값은 매 구역이 발견된 회수이다. END블로크는 특수한 awk for순환을 사용하여 region이라는 배렬을 반복한다.

UNIX 편의프로그람에 대한 련습

련습문제 6: nawk련습

Mike Hamneion: (510.). 548-1278:250:100 175
Christian Dobbins:(408.). 538-2358 155.90:201
Susan Dalsass:(206.). 654-6279 250 60:50
Archie McNichol:(206.). 548-1348 :250:100-175
Jody Savage:(206.).548-1278:15:188:150
Guy Quiglo (916.).343-6410 250 100 175
Dan Saverage(206.). 298-7744:450:300:275
Nancy McNeil(206.). 548-1278:250:100:175
John Golderod:(916.).348-4278:250:100:175
Chet Main:(510.).548-5258:50:95:135
Tom savage:(408.).926-3456:250:168:200
Elizabeth Stachelin:(916.).440-1763:175:75:300

(CD의 lab6.data라는 자료기지를 참조하시오.).
우의 자료기지에는 이름, 전화번호, 석달에 걸쳐 진행한 기부금들이 있다.

1. 다음의 보고서를 제출하는 nawk스크립트를 쓰시오.

FIRST QUARTERLY REPORT

CAMPAIGN 1998 CONTRIBUTIONS

NAME		PHONE	Jan	Fev	Mar	TotalDonated
Mike Harrington	(510)	548-1278	250.00	100.00	175.00	525.00
Christian Dobbins	(408)	538-2358	155.00	90.00	201.00	446.00
Susan Dalsass	(206)	654-6279	50.00	60.00	50.00	360.00
Archie McNichol	(206)	548-1348	250.00	100.00	175.00	525.00
Jody Savage	(206)	548-1278	15.00	188.00	150.00	353.00
Guy Quigley-	(916)	343-6410	250.00	100.00	175.00	525.00
Dan Savage--	(406)	298-7744	450.00	300.00	275.00	1025.00
Nancy McNeil	(206)	548-1278	250.00	80.00	75.00	405.00
John Goldenrod	(916)	348-4278	250.00	100.00	175.00	525.00
Chet Main	(510)	548-5258	50.00	95.00	135.00	280.00
Tom Savage	(408)	926-3456	250.00	168.00	200.00	618.00
Elizabeth Stacbelin	(406)	440-1763	175.00	75.00	300.00	550.00

```
SUMMARY
-------------------------------------------------------------------------------
The campaign received a total of $6137.00 for this quarter.
The average donation for the 12 contributors was $511.42.
   The highest totalcontribution was $1025.00 made by Dan Savage.
                              ***THANKS Dan***
   The following people donated over $500 to the campaign.
   They are eligible for the quarterly drawing!!
   Listed are their names (sorted by last names) and phone numbers:
         John Goldenrod--(916.). 348-4278
         Mike Harnngton--( 510) 548-1278
         Archie McNichol--(206) 548-1348
         Guy Quigley--(916) 343-6410
         Dan Savage--(406) 298-7744
         Tom Savage--(408) 926-3456
         Eizabeth Stachelin--(916) 440-1763
                   Thanks to all of you for ybur continued support!!
```

제 14 절. 기타 문제들

일부 자료(실레로 테프 등으로부터 읽은 자료)는 명백한 마당분리기호를 가지지 않고 그대신에 고착된 너비의 렬들을 가질수 있다. 이 자료형들을 먼저 처리하는데 substr함수가 유용하다.

1. 고정마당

다음의 실레에서는 마당이 고정된 폭을 가지지만 마당분리기호들에 의해 분리되지 않는다. substr함수는 마당들을 작성하는데 리용된다.

실례 7-81

```
% cat fixed
031291ax5633(408.).987-0124
021589bg2435(415)866-1345
122490del237(916)933-1234
010187ax3458(408)264-2546
092491bd9923(415)134-8900
112990bg4567(803)234-1456
070489gr3455(415)899-1426
% nawk '{printf substr($0,l,6)" ";prlntf substr ($0, 7, 6) " ";\
print substr($0,13,length.).}' fixed
031291   ax5633   (408.).987-0124
021589   bg2435   (415.).866-1345
122490   del237   (916.).933-1234
```

```
010187   ax3458   (408.).264-2546
092491   b(S9923   (415.).134-8900
112990   bg4567   (803.).234-1456
070489   qr3455   (415.).899-1426
```

설명

첫 마당은 전체 레코드의 부분문자렬을 첫 문자렬로부터 시작하여 6자리를 취하여 얻는다. 다음에 공백이 인쇄된다. 두번째 마당은 일곱번째 위치로부터 시작하여 6개의 자리를 취하고 그뒤에 공백을 넣어 얻는다. 마지막마당은 열세번째 위치로부터 시작하여 행의 길이에 의해 표현되는 위치까지 전체 레코드의 부분문자렬을 취해 얻는다(길이함수는 인수를 가지지 않으면 현재행($0)의 길이를 되돌려 준다.).

NULL마당 자료가 고정된 너비의 마당에 기억되면 그 일부가 NULL마당으로 될수 있다. 다음의 실례에서는 substr함수를 리용하여 자료를 포함하든 관계없이 마당을 보존한다.

실례 7-82

```
1.          % cat db
xxx xxx
xxx abc xxx
xxx a   bbb
xxx       xx
% cat awkfix
# Preserving empty fields. Field width is fixed.
{
2.          f[1]=substr($0,1,3)
3.          f[2]=substr($0,5,3)
4.          f[3]=substr($0,9,3)
5.          line=sprintf("%-4s%-4s%-4s\ n",f[1],f[2],f[3].).
6.          print line
}
% nawk –f awkfix db
xxx xxx
xxx abc xxx
xxx a    bbb
xxx        xx
```

설명

1. 파일 db의 내용이 인쇄된다. 파일에는 NULL마당들이 있다.
2. f배렬의 첫번째 요소는 레코드의 위치 1에서 시작하여 3개의 자리수를 가지는 부분문자렬이 대입된다.
3. f배렬의 두번째 요소에는 레코드의 위치 3에서 시작하여 5개의 자리수를 가지는 부분문자렬이 대입된다.
4. f배렬의 세번째 요소에는 레코드의 위치 9에서 시작하여 3개의 자리수를 가

지는 부분문자렬이 대입된다.

5. 배렬요소는 sprintf함수에 의해 형식화된후에 사용자정의변수 line에 대입된다.
6. Line의 값이 인쇄되고 NULL마당들은 보존된다.

$, 반점, 기타 문자를 가지는 수 다음의 실례에서는 일부 마당이 화페문자 $와 반점(,)을 가진다. 스크립트는 가격들을 합계하여 전체 가격을 얻기 위해 이 문자들을 지운다. 이것은 gsub함수를 사용하여 수행된다.

실례 7-83

```
% cat vendor
access tech:gp237221:220:vax789:20/20.-n/OI/90:$-!< 043. 00
alisa systems :bp262292 : 280 :macint:osh:rev -:'y;a':es: 06/30/91: $456 . 00
alisa systems:gp262345:260:vax8700:alias talk:02/03/91: $1, 598. 50
apple computer: zx342567 : 240 :macs: e-mail: 06/25/9090: $575.75
cac;I;gp2ff23J' 3:2 80: spare station :network1.5:05/12/91/:$1,250.75
datalogics:bpl32455:260:microvax2:pagestation maint:07/01/90:$1,200.00
dec:zx354612:220:microvax2:vms sms:07/20/90:$1,350.00
% nawk –F: '{gsub(/\  $/," ".).;gsub(/,/,-" ".).; cost +=-$7};\
        END{print "The total is $" cost}' vendor
$7474
```

설명

첫번째 gsub함수는 화페기호(\ $)를 빈 문자렬로 교체하고 두번째 gsub함수는 반점(,)을 빈 문자렬로 교체한다. 사용자정의변수 cost에 일곱번째 마당을 더하고 그 결과값을 다시 cost에 대입하여 그 값을 합계한다. END블로크에서 문자렬 The total cost is $가 인쇄되고 다음 cost[1]의 값이 인쇄된다.

2. 파일묶기(Bundling)와 풀기(Unbundling)

묶기프로그람 awk프로그람작성언어에서 파일을 묶는 프로그람은 대단히 간단하다. 디스크공간을 절약하고 전자우편을 통하여 파일을 보내기 위하여 여러개의 파일을 한개의 파일로 결합한다. 다음의 awk지령은 파일이름이 앞에 놓인 매 파일의 모든 행들을 인쇄한다.

실례 7-84

```
% nawk '{ print FILENAME, $0 }' file1 fiel2 file3 > boundled
```

---

[1]. commas 가 어떻게 프로그람에 추가되는가를 구체적으로 보기 위해서는 Alfred V.Aho, Brian W. Kernighan, and Peter J.Weinberger가 쓴 AWK프로그람작성언어(Boston:Addision-Wesley, 1988)의 72페지를 보시오..

설명

현재입력파일의 이름 FILENAME이 인쇄되고 그뒤에 file1의 매 입력행에 대하여 레코드($0)를 인쇄한다. file1이 파일의 끝을 탐색한후 awk는 다음의 파일 file2를 열고 우와 같은 동작을 반복한다. 출력은 bundled라는 파일로 방향바꾸기된다.

풀기프로그람 다음의 실례는 파일들을 어떻게 푸는가 혹은 개별 파일들로 만드는가를 표시한다.

실례 7-85

```
% nawk `$1 != previous { close(previous.).; previous};\
{ print subatr($0,index($0, index($0," ".).+1.). > $1.).` boundled
```

설명

첫 마당은 파일의 이름이다. 파일의 이름이 사용자정의변수 previous(처음에 빈값)의 값과 같지 않으면 동작블로크가 실행된다. previous에 대입한 파일은 닫기고 previous는 첫 마당값으로 대입된다. 다음에 레코드의 substr 즉 첨수함수로서 되돌려 지는 시작위치(첫 공간+1의 위치)는 첫 마당에 포함되는 파일이름으로 방향바꾸기된다. 파일이름이 한행에 단독으로 나타나도록 파일들을 묶기 위해 다음의 지령을 사용한다.

```
% nawk c{if (FNR= =1.).{print FILENAME;print $0}\
else print $0}'   file1   file2   file3   > bundled
```

다음의 지령은 파일들을 다시 푼다.

```
% nawk NF= =1{filename= ;\
NF ! = 1 {print   $0   >filename} '    bundled
```

3. 여러행레코드

앞에서 사용된 표본자료파일에서 매 레코드는 단독으로 한행우에 놓인다. checkbook라는 다음의 표본자료파일에서 레코드들은 공백행으로, 마당들은 행바꾸기로 구별된다. 이 파일을 처리하기 위해 레코드분리기호(RS)에 빈값이, 마당분리기호(FS)에는 행바꾸기가 대입된다.

실례 7-86

(The Input File.).

% cat cbeckbook

1/1 01

#125

-695.00

Mortgage

1/1/01

```
#126
-56.89
PG&E
#127
-89.99
Safeway
1/3/01
+750.00
Pay Check

1/4/01
#128
-60.00
Visa

(The Script.).
% cat awkchecker
1. BEGIKI{RS=" " ; FS="\ n";ORS="\ n\ n"}
2. {print   NR, $1,$2,$3,$4}
(The Output.).
% nawk –f awkchecker checkbook
1 1/1/01   #125   -695.00   Mortgage
2 1/1/01   #126   -56.89   PG&E
3 1/2/01   #127   -89.99   Safeway
4 1/3/01   +750.00   Pay Check
5 1/4/01   #128   -60 00   Visa
```

설명

1. BEGIN블로크에서 레코드분리기호에 빈값이, 마당분리기호(FS)에 행바꾸기가, 레코드분리기호(ORS)에는 2개의 행바꾸기가 대입된다. 현재 매행은 한개의 마당으로 되며 매 출력레코드는 2개의 행바꾸기에 의해 분리된다.
2. 레코드번호를 인쇄하고 그뒤에 모든 마당들을 인쇄한다.

4. 형식글자발생

다음의 실례는 AWK프로그람작성언어[2]의 프로그람으로부터 변경할수 있다. 여기서

[2] Alfred V.Aho, Brian W.Kernighan, Peter J.Weinberger, AWK 프로그람작성언어 (Boston: Addson-Wesley, 1988).©1988 Bell Telephone Laboratories, Inc.Reprinted by permission of Pearson Education, Inc.

기교적인 부분은 무엇이 실제적으로 처리되고 있는가를 아는데 있다. 입력파일을 data.form이라고 하였다. 여기에는 바로 자료가 포함되여 있다. 입력파일에서 매 마당은 두점으로 분리된다. 다른 파일은 form.letter라고 한다. 이것은 형식문자를 만드는데 사용되는 실제적인 형식이다. 이 파일은 getline함수에 의해 awk기억기에 넣어 진다. 형식글자의 매행은 배렬에 기억된다. 프로그람은 data.form으로부터 자료를 얻고 문자는 form.letter에 있는 #와 @가 앞에 붙은 특수문자렬들을 실지자료로 교체하여 작성한다. 림시변수 temp는 자료가 교체된후에 표시되는 실제적인 행을 유지한다. 이 프로그람은 data.form에서 렬거되는 매 사람들에 대하여 개별화된 형식문자들을 작성할수 있게 한다.

실례 7-87

```
(The Awk Script)
% cat form.awk
# form.awk is script that requires access to 2 files: The
# first file is called "This file contains the
# format for a form letter. The awk script uses another file,
# "data.form," as its input file. This file contain the
# information that will be substituted into the form letters in
# the place of the numbers preceded by pound signs. Today's date
# is substituyed in the place of "@date" in "form.letter."
1   BEGIN{ FS=":"; n=1
2   while ( getline < "form.letter" > 0 )
3         form[n++] = $0      # store lines from form.letter in an array
4   "date" | getline d; split (d, today, " ")
          # Output of date is Fri Mar 2 14:35:50     PST 2001
5   thisday=today[2]"."today[6]
6   }
7   { for ( i = 1 ; j < n ; i++ ) {
8         temp=form[i]
9         for ( j = 1 ; j <= NF ; j++ ) {
              gsub ( "@date", thisday, temp )
10            gsub ( "#" j, $1, temp )
          }
11    print temp
          }
          }

% cat form.letter
The form letter, form.letter,looks like this:
```

```
***********************************************************
    subject:Status Report for Project "#1"
    To:   #2
    From: #3
    Date: @date
    This letter is to tell you, #2, that project "#1" is uo tp
    Date.
    We expect that everything will be completsd and ready for
    shipmaent as scheduled on 4.

    Sincerely,

    #3
***********************************************************
```

The file, *data. Form,* is awk's **input file** containing the data that will replace the #1-4 and the @date in *form.letter.*

```
% cat data.form
    Dynamo:John Stevens:Dana Smith, Mgr:4/12/2001
    Gallactius:Guy Sterling:Dana Smith. Ngr:5/l8/2001
```

(The Command Line.).

```
    % nawk   -t form.awk   data.form
***********************************************************
```

Subject: Status Report for Project. "Dynamo"
To: John Stevens
From: Dana Smi th, Mgr
Date: Mar. 2, 2001
This letter is to tell you, John Stevens, that project
"Dynamo" Is up to date.
We expect that everything will be completed for shipment as scheduled, on 4/12/2001.

Sincerely,

Dana Smith, Mgr Subject: Status Report for Project   "Gallactius"
To: Guy Sterling
From: Dana Smith, Mgr
Date: Mar. 2, 2001
This letter is to you. Guy Sterling, that project 'Gallactius'
is up to date.
We expect that everything will be completed and   ready for
shipment as scheduled on 5/18/2001.

Sincerely,

Dana Smith, Mgr

설명

1. BEGIN블로크에서 마당분리기호(FS)는 두점으로 대입된다. 사용자정의변수 n은 1로 대입된다.
2. While순환에서 getline함수는 form.letter 파일로부터 한번에 한행을 읽는다. getline함수가 파일찾기를 실패하면 -1을, 파일의 끝을 탐색하면 0을 되돌려 준다. 따라서 되돌이값이 1보다 큰가를 검사하여 함수가 입력파일로부터 한행을 읽었다는것을 알수 있다.
3. Form.letter로부터 읽는 매행은 form이라는 배렬에 대입된다.
4. UNIX date지령으로부터의 출력은 getline함수에 파이프되고 사용자정의변수 d에 대입된다. 그다음 split함수는 today라는 배렬을 작성하면서 변수 d를 공백으로 분리한다.
5. 사용자정의변수 this day에 월, 날자, 년이 대입된다.
6. BEGIN블로크는 끝난다.
7. for순환은 n번 순환한다.
8. 사용자정의변수 temp는 form배렬로부터 한행을 대입 받는다.
9. 겹for순환은 입력파일 data.form으로부터 NF수만큼 한행씩 순환한다. Temp변수에 기억된 매행은 문자렬 @date를 검사한다. @date가 일치되면 gsub함수는 그것을 오늘의 날자로 바꾼다(this day에 기억된 값).
10. temp에 기억되는 행에서 #와 수를 발견하면 gsub함수는 #와 수를 입력파일 data.form에 대응하는 마당의 값으로 교체한다. 실례로 기억된 첫 행이 검사되고 있으면 #1은 Dynamo로, #2는 Jhon Stevens, #3은 Dana Smith로, #4는 4/12/2001 등으로 바뀐다.
11. Temp에 기억된 행은 교체후에 인쇄된다.

5. 쉘과의 대화

지금까지 awk가 어떻게 동작하는가를 보았으므로 awk가 쉘스크립트를 작성할 때 아주 강력한 편의프로그람이라는것을 알수 있다. 사용자는 쉘스크립트내에서 한행에 놓이는 awk지령이나 awk스크립트를 포함할수 있다. 다음의 실례는 awk지령들을 가진 Korn쉘프로그람의 실례이다.

실례 7-88

```
!#/bin/ksh
# This korn shell script will collect data for awk to use in
# generating form letter is.).. See above.
Print “Hello $LOGNAME. “
print “This report is for the month and year:”
1.  cal | nawk ‘HR==l{print $0}’
    if [[ -f data.form     | -f formletter-? ]]
    then
            rm data.form formletter’5    2> /dev/null
```

```
        fi
        integer num=l
        while true
        do
        print "Form letter #$num:"
        read project?"What is the name of the project? "
        read sender?"Who is the status report from'5 "
        read recipient?"Who is the status report to7 "
        read due_date?"What is the completion date scheduled7
        echo $project:$recipient:$sender:$due_date > data.form
        print –n "Do you wish to generate another form letter?
        read answer
        if [[ "$answer" ‘:= [Yy]* ]]
        then
                break
        else
2.              nawk –f   fonn.awk   data.form   > formletter$num
        fi
        (( num+=l .)..).
done
nawk –f form.awk data. Form > formletfcer$nai
```

설명

1. UNIX cal지령은 awk에로 파이프된다. 현재의 달과 년을 포함하는 첫 행이 인쇄된다. nawk스크립트 form.awk는 UNIX파일들에로 방향바꾸기되는 형식문자들을 만든다.

제 15 절. 개괄

1. 문자렬함수

실례 7-89

```
% nawk `NR==1{gsub(/northwest/,"southeast", $1.). ;print}` datafile
southeast     NW     Joel Craig        3.0     .98        3   4
```

설명

이것이 첫번째 레코드(NR==1)일 때 northwest가 첫번째 마당에 있으면 그것을 southeast로 모두 치환한다.

```
% cat datafile
northwest    NW    Joel Craig       3.0   .98   3    4
western      WE    Sharon Kelly     5.3   .97   5   23
southwest    SW    Chris Foster     2.7   .8    2   18
southern     SO    May Chin         5.1   .95   4   15
southeast    SE    Derek Johnson    4.0   .7    4   17
eastern      EA    Susan Beal       4.4   .84   5   20
northeast    NE    TJ Nichols       5.1   .94   3   13
north        NO    Val Shultz       4.5   .89   5    9
central      CY    Sheri Watson     5.7   .94   5   13
```

실례 7-90

```
% nawk `NR==1{print substr($3, 1, 3)}` datafile
Joe
```

설명

이것이 첫 레코드이면 첫 문자로 시작하여 3개의 문자로 이루어 진 세번째 마당의 부분문자렬을 표시한다. 부분문자렬 Joe가 인쇄된다.

실례 7-91

```
% nawk `NR==1{print length($1)}` datafile
9
```

설명

이것이 첫 레코드이면 첫 마당의 길이(문자수)가 표시된다.

실례 7-92

```
% nawk `NR==1{print index($1,"west".).}`   datafile
6
```

설명

이것이 첫 레코드이면 첫 마당에서 부분문자렬 west가 놓이는 첫번째 위치를 인쇄한다. 문자렬 west는 문자렬 northwest의 여섯번째 위치(index)에서 시작된다.

실례 7-93

```
% nawk `if(match($1,/^no/.)..). {print substr($1,RSART,RLENGTH)}}`\
```

datafile

no

no
no

설명

match함수가 정규식 /^no/을 첫 마당에서 찾으면 제일 왼쪽 문자의 첨수위치를 되돌려 준다. 내부변수 RSTART는 첨수위치로 설정되며 RLENGTH변수는 일치된 부분문자렬길이로 설정된다. substr함수는 첫번째 마당에서 위치 RSTART에서 시작되면서 RLENGTH의 문자수만한 길이를 가지는 문자렬을 되돌려 준다.

실례 7-94

% **nawk `BEGIN{split("10/14/01",now,"/").;print now[1],now[2],now[3]}'**
10 14 01

설명

문자렬 10/14/01은 now라는 배렬로 나누어 진다. 분리기호는 빗선기호이다. 배렬의 요소들은 배렬의 첫 요소부터 인쇄된다.

% cat datafile2

```
Joel Craig:northwest:          NW: 3.0 : .98 : 3 : 4
Sharon Kelly:western:          WE: 5.3 : .97 : 5 : 23
Chris Foster:southwest:        SW: 2.7 : .8  : 2 : 18
May Chin:southern:             SO: 5.1 : .95 : 4 : 15
Derek Johnson:southeast:       SE: 4.0 : .7  : 4 : 17
Susan Beal:eastern:            EA: 4.4 : .84 : 5 : 20
TJ Nichols:northeast:          NE: 5.1 : .94 : 3 : 13
Val hultz:north:               NO: 4.5 : .89 : 5 : 9
Sheri Watson:central:          CY: 5.7 : .94 : 5 : 13
```

실례 7-95

% **nawk –F: '/north/{split($1,name, " ").;**
print "First name: "name[1];
print "Last name: "name[2];
print "\ n- - - - - - - - -"}' datafile2

First name: Joel

```
last name: Craig
---------------------------------
First name: TJ
last name: Nichols
---------------------------------
First name: Val
last name: Shultz
---------------------------------
```

설명

입력마당분리기호는 두점(-F:)으로 설정된다. 레코드가 정규식 north를 포함하면 첫 마당은 name이라는 배렬로 나누어 지는데 이 배렬에서 공백은 분리기호이다. 배렬요소들이 인쇄된다.

% cat datafile

northwest	NW	Joel Craig	3.0	.98	3	4
western	WE	Sharon Kelly	5.3	.97	5	23
southwest	SW	Chris Foster	2.7	.8	2	18
southern	SO	May Chin	5.1	.95	4	15
southeast	SE	Derek Johnson	4.0	.7	4	17
eastern	EA	Susan Beal	4.4	.84	5	20
northeast	NE	TJ Nichols	5.1	.94	3	13
north	NO	Val Shultz	4.5	.89	5	9
central	CY	Sheri Watson	5.7	.94	5	13

실례 7-96

```
% nawk '{line=sprintf("%10.2f%5s\ n",$7,$2).; print line}' datafile
3.00   NW
 5.00   WE
 2.00   SW
 4.00   SO
4.00   SE
5.00   EA
3.00   NE
5.00   NO
5.00   CT
```

설명

sprintf함수는 printf함수의 형식규칙을 사용하여 일곱번째 마당과 두번째 마당($7, $2)을 형식화한다. 형식화된 문자렬은 되돌려 지고 사용자정의변수 line에 대입되여 인쇄된다.

2. 지령행인수

실례 7-97

```
% cat argvs.sc
# Testing command line arguments with ARGV and ARGC using a for loop.

BEGIN{
   for(i=0;i < ARGC;i++)
        printf("argv[%d] is %s\ n", i, ARGV[I])
        printf("The number of arguments. ARGV=%d\ n", ARGC.).
}

% nawk -f argvs.se datafile
argv[0] is nawk
argvil] is datafile
The number of arguments, ARGC=2
```

설명

BEGIN블로크는 for순환을 가지고 지령행인수들을 처리한다. ARGC는 인수개수이고 ARGV는 실제인수들을 가진 배렬이다. nawk는 추가선택들을 인수로 계수하지 않는다. 이 실례에서 유효한 인수들은 다만 nawk지령과 입력파일인 datafile뿐이다.

실례 7-98

```
% nawk 'BEGIN{name=ARGV[1]};\
  $0 ~name {print $3, $4}'   "Derek" datafile
   nawk: can't open Dereck
   source line number 1
   % nawk 'BEGIN{name=ARGV[1]=ARGV[1]; delete ARGV[1]};\
      $0 ~ name {print $3, $4}' "Dreck" datafile
Dreck Johnson
```

설명

1. 이름 “Deck”는 BEGIN블로크에서 변수 name으로 설정되였다. 패턴동작블로크에서 nawk는 입력파일로써 “Derek”를 열려고 하다가 실패하였다.

2. "Derek"를 변수 name에 대입하면 ARGV[1]이 지워 진다. 패턴동작블로크를 시작할 때 nawk는 입력파일로써 "Derek"대신에 datafile을 연다.

% cat datafile

northwest	NW	Joel Craig	3.0	.98	3	4
western	WE	Sharon Kelly	5.3	.97	5	23
southwest	SW	Chris Foster	2.7	.8	2	18
southern	SO	May Chin	5.1	.95	4	15
southeast	SE	Derek Johnson	4.0	.7	4	17
eastern	EA	Susan Beal	4.4	.84	5	20
northeast	NE	TJ Nichols	5.1	.94	3	13
north	NO	Val Shultz	4.5	.89	5	9
central	CY	Sheri Watson	5.7	.94	5	13

3. 입력읽기

실례 7-99

% **nawk 'BEGIN{ "date" | getline d}; ;print d}' datafile**

Mon Jan 11:24:24 PST 2001

설명

UNIX date지령은 getline함수에 파이프된다. 결과는 변수 d에 기억되고 인쇄된다.

실례 7-100

% **nawk 'BEGIN{ "date" | getline d; split(d, mon) :print mon[2]}**

datafile

Jan

설명

UNIX date지령은 getline함수에 파이프되며 결과는 변수 d에 기억된다. split 함수는 문자렬 d를 mon이라는 배렬로 나눈다. 두번째 배렬요소가 인쇄된다.

실례 7-101

```
% nawk 'BEGIN{ print "who are you looking for?" ;\
getline name < "/dev/tty"};\
```

설명

입력은 말단 /dev/tty에서 읽기하여 name이라는 배렬에 기억시킨다.

실례 7-102

```
% nawk 'BEGIN{ while(getline < "/etc/passwd" > 0) {lc++}; print lc}'\
datafile
16
```

설명

while순환은 한번에 한행씩 /etc/passwd파일을 순환하는데 리용된다. 순환에 들어 갈 때마다 getline에 의해 행을 읽으며 변수 lc의 값은 증가된다. 순환에서 탈퇴할 때 lc의 값이 인쇄된다. 즉 /etc/passwd파일에서 행번호들이 인쇄된다. getline의 되돌이 값이 0이 아니면 즉 행이 읽어 졌으면 순환이 계속된다.

4. 조종함수

실례 7-103

```
% nawk '{if  ( $5 > 4.5) next; print $1}'  datafile
northwest
southwest
southeast
eastern
north
```

설명

네번째 마당이 4.5보다 크면 입력파일(datafile)로부터 다음행을 읽고 awk스크립트의 머리부(BEGIN블로크다음)에서 처리가 시작된다. 그렇지 않으면 첫번째 마당이 인쇄된다.

실례 7-104

```
% nawk '{if ($2 ~ /s/.). {print ; exit 0}}' datafile
southwest     SW    Chris    Foster    2.7    .8    2    18
```

```
% echo $status ( csh .). or echo $? (sh or ksh)
0
```

설명

두번째 마당이 S를 가지고 있으면 그 레코드가 인쇄되고 awk프로그램은 탈퇴한다. C쉘상태변수는 탈퇴값을 가진다. 만일 Bourne나 Korn쉘을 사용하면 $?변수는 탈퇴상태를 가진다.

% cat datafile

northwest	NW	Joel Craig	3.0	.98	3	4
western	WE	Sharon Kelly	5.3	.97	5	23
southwest	SW	Chris Foster	2.7	.8	2	18
southern	SO	May Chin	5.1	.95	4	15
southeast	SE	Derek Johnson	4.0	.7	4	17
eastern	EA	Susan Beal	4.4	.84	5	20
northeast	NE	TJ Nichols	5.1	.94	3	13
north	NO	Val Shultz	4.5	.89	5	9
central	CY	Sheri Watson	5.7	.94	5	13

5. 사용자정의함수

실례 7-105

(The Command Line.).

```
% cat nawk.sc7
1.   BEGIN{largest=0}
2.   (maximum=max($5).).
3.   function max ( num .). {
4.         if ( num > largest){ largest=num }
         return largest
5.   }
6.   END{ print "The maximum is " maximum "."}

% nawk -f nawk.sc? datafile

The maximum is 5.7.
```

설명

1. BEGIN블로크에서 사용자정의변수 largest가 령으로 초기화된다.
2. 파일의 매행에서 변수 maximum은 함수 max로부터 되돌려 지는 값을 가진다. 함수 max에 $5가 인수로 주어 진다.
3. 사용자정의함수 max가 정의된다. 함수명령문은 대괄호안에 놓인다. 새 레코드가 입력파일인 datafile로부터 읽어 질 때마다 함수 max가 호출된다.
4. num과 largest에서 값들을 비교하여 큰 수를 되돌려 준다.
5. 함수정의블로크가 끝난다.
6. END블로크가 maximum의 마지막값을 인쇄한다.

UNIX편의프로그람련습

련습문제 7: nawk련습

Mike Hamneion: (510.). 548-1278:250:100 175

Christian Dobbins:(408.). 538-2358 155.90:201

Susan Dalsass:(206.). 654-6279 250 60:50

Archie McNichol:(206.). 548-1348 :250:100-175

Jody Savage:(206.).548-1278:15:188:150

Guy Quiglo (916.).343-6410 250 100 175

Dan Saverage(206.). 298-7744:450:300:275

Nancy McNeil(206.). 548-1278:250:100:175

John Golderod:(916.).348-4278:250:100:175

Chet Main:(510.).548-5258:50:95:135

Tom savage:(408.).926-3456:250:168:200

Ehzaeth Stachelin:(916.).440-1763:175:75:300

(CD에서 lab7.data라는 자료기지를 참조하시오.)

자료기지는 이름, 전화번호, 석달에 걸치는 기부금이 있다.

1. 주어 진 달에서 모든 기부금을 평균값으로 되돌려 주는 사용자정의함수를 쓰시오. 달은 지령행에서 통과(무시)될수 있다.

제 8 장. 대화형Bourne쉘

제 1 절. 기 동

Bourne쉘이 등록가입쉘이라면 쉘재촉문이 나타나기전에 몇개의 프로세스들이 련속적으로 실행된다.

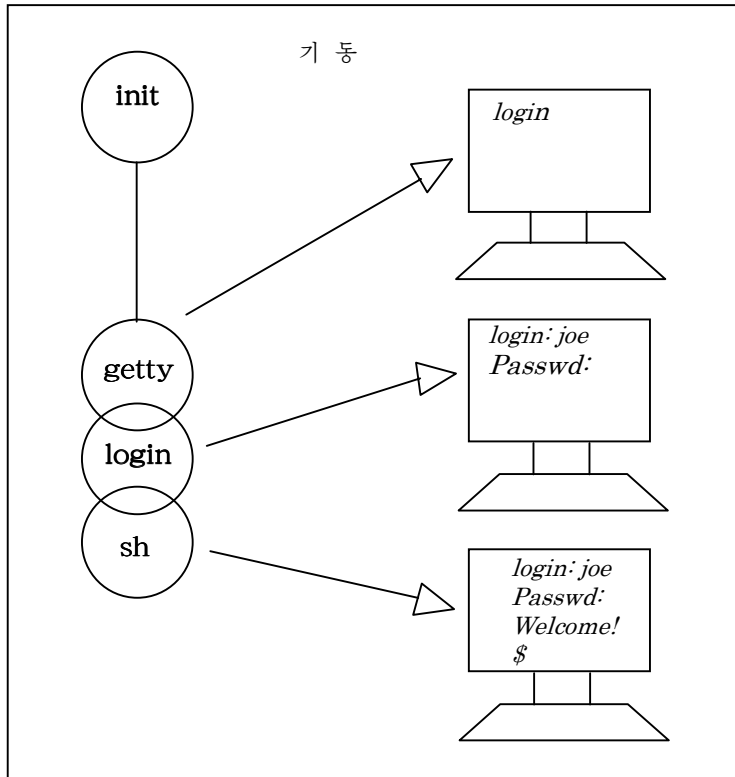



그림 8-1. Bourne 의 기동

처음으로 실행되는 프로세스는 PID가 1인 init이다. 이것은 inittab(system V) 파일로부터 명령을 받는다. 즉 getty프로세스(BSD)를 파생시킨다. 이 프로세스들은 말단포구들을 열어 표준입력, 표준출력, 표준오유를 제공해 주며 현시장치에 login재촉문을 현시해 준다. login프로그람은 암호를 재촉하고 그것을 해득하여 확인한 다음 초기환경을 설정하며 passwd파일에 있는 마지막입구점인 등록가입쉘 /bin/sh를 기동시킨다. sh프로세스는 체계파일 /etc/profile을 찾아서 그의 지령들을 실행한다. 그다음 .profile이라고 하는 초기화파일을 사용자의 홈등록부에서 찾는다. profile로부터 지령들이 실행된후 기정화페기호($)재촉문이 현시장치에 나타나고 Bourne쉘이 지령을 대기한다.

1. 환경

프로세스의 환경은 변수, 열린파일, 현재작업등록부, 함수, 자원한계, 신호 등으로 이루어 진다. 환경은 한 쉘에서 다음쉘로 계승되는 특징들과 작업환경에 대한 구성을 정의한다. 사용자쉘에 대한 구성은 쉘초기화파일에서 정의된다.

초기화파일 Bourne쉘프로그람은 기동후 체계파일인 /etc/profile을 찾는다. 이 파일의 지령들이 실행된후 사용자홈등록부에 있는 초기화파일인 .profile이 실행된다. 그 다음 /bin/login프로그람이 실행된다. 초기설정을 위한 구조파일들은 /etc/skel(SVR4)에서 찾을수 있다.

/etc/profile파일 /etc/profile파일은 사용자가 등록가입할 때 과제들을 수행하는 체계관리자에 의하여 설정되는 전 체계적인 초기화파일이다. 이것은 Bourne쉘이 기동할 때 실행된다. 이 파일은 체계우의 모든 Bourne쉘과 Korn쉘사용자들이 사용할수 있으며 새로운 우편을 위하여 우편완충기를 검사하거나 /etc/motd파일로부터 그날의 통보를 현시하는 것과 같은 과제를 수행한다(아래의 실례들은 이 장의 학습을 끝내면 더 알기 쉽다.).

실례 8-1

```
(Sample/etc/profile)
# The profile that all logins get before using their own .profile
1.  trap " " 2 3
2.  export LONGNAME PATH
3.  if [ "$TERM" = " " ]
    then
        if /bin/i38.6
        then
            TERM=AT38.6     # Sets the terminal
        else
            TERM=sun
        fi
        export TERM
    fi
    # Login and -su shells get /etc/pro file services.
    # -rsh is given its environment in its own .profile.
4.    case "$0" in
      -sh | -ksh | -jsh )
5.        if [ ! -f .hushlogin ]
          then
              /usr/sbin/quota
              #Allow the user to break the Message-Of-The-
              # Day only.
6.            trap "trap ' ' 2" 2
7.            /bin/cat -s /etc/motd
             # Message of the day displayed.
               trap " " 2
8.             /bin/mail -E        # Checks for new mail
9.             case $? in
```

```
                        0)
                              echo "You have new mail. "
                        ;  ;
                        2)
                              echo "You have mail.
                              ; ;
                        esac
                fi
          esac
10.       umask 022
11.       trap 2 3
```

설명

1. trap지령은 프로그람이 실행되는 동안 전송되는 신호들을 조종한다. 만일 프로그람이 실행상태에 있을 때 신호 2(Control-C)나 3(Control-\ )이 전송되면 신호들은 무시된다.
2. 변수 LOGNAME과 PATH가 넘겨 져 그 값들이 프로쎄스에서 파생된 보조쉘에 알려 진다.
3. 지령 /bin/i386이 실행된다. 지령의 탈퇴상태가 령이면 말단변수 TERM은 값이 AT386으로 된다. 령이 아니면 TERM변수에 sun이 대입된다.
4. /etc/profile파일을 실행시키는 프로그람의 이름인 $0의 값은 등록가입쉘, Bourne쉘, Korn 쉘 또는 작업쉘이며 다음지령이 실행된다.
5. .hushlogin파일이 존재하지 않을 때 quotas가 실행되여 디스크사용경고를 현시한다.
6. 사용자가 그날의 통보를 Control-C로 끝낼수 있도록 trap가 재설정된다.
7. 그날의 통보가 현시된후 trap가 재설정되여 Control-C를 무시한다.
8. 우편프로그람이 새로 들어 오는 우편을 검사한다.
9. 우편프로그람의 탈퇴상태($?)가 0 또는 2이면 통보"You have new mail", "You have new mail" 또는 "You have mail"이 현시된다.
10. umask지령이 설정되여 파일들과 등록부들이 작성될 때 그것들의 초기화허가를 결정한다.
11. trap지령은 신호 2와 3을 다시 기정값으로 설정한다. 즉 Control-C나 Control- \ 이 도착하면 프로그람을 중지시킨다.

.profile파일 profile파일은 등록가입할 때 한번 실행되는 사용자정의초기화파일이며 홈등록부에 놓인다. 이것은 쉘환경을 사용자의 요구에 맞게 수정할수 있게 한다.

환경과 말단설정들은 보통 이 파일에 있는데 windows응용프로그람이나 자료기지응용프로그람은 초기화되면 여기서 시작된다. 이 파일에 있는 설정들은 뒤에서 자세히 설명되는데 여기서는 파일의 매행에 대하여 간단히 설명한다.

실례 8-2

(Sample *.profile*)

```
1.   TERM=vtl02
2.   HOSTMAME='uname -n'
```

```
 3.  EDITOR=/usr/ucb/vi
 4.  PATH=/bin:/usr/ucb:/usr/bin:/usr/local:/etc:/bin:/usr/bin:
 5.  PS1="$HOSTNAME $ > "
 6.  export TERM HOSTNAME EDITOR PATH PS1
 7.  stty erase ^h
 8.  go () { cd $1; PS1='pwd'; PS1='baeenaine $PS1'; }
 9.  trap '$HOME/.logout' EXIT
10.  clear
```

설명

1. TERM변수의 값은 말단형태의 값 Vt102로 된다.
2. uname-n지령이 거꿀인용부호안에 있기때문에 쉘은 지령바꿔넣기를 한다. 즉 지령의 출력(주콤퓨터의 이름)은 변수 HOSTNAME에 대입된다.
3. EDITOR변수는 /usr/ucb/vi로 대입된다. mail과 같은 프로그람은 편집기를 정의할 때 이 변수를 쓸수 있게 한다.
4. PATH변수에는 쉘이 UNIX프로그람을 찾기 위해 탐색하는 등록부입구점들이 대입된다. 실례로 rs를 입력하면 쉘은 목록화된 등록부들에서 프로그람을 찾을 때까지 PATH변수를 탐색한다. 프로그람을 찾지 못하면 쉘은 그것을 통보한다.
5. 1차재촉문에 콤퓨터이름 HOSTNAME, $와 >부호들이 대입된다.
6. 모든 목록화된 변수들을 넘긴다. 그것들은 이 쉘에서부터 파생되는 자식프로쎄스들에 알려 진다.
7. stty지령은 말단추가선택들을 설정한다. 지우기건을 ^h로 설정하고 backspace건을 누를 때 유표앞에 있는 문자가 지워 진다.
8. go라고 하는 함수를 정의한다. 이 함수의 목적은 한개의 인수인 등록부이름을 받아서 등록부를 그 등록부로 변환시키고 초기재촉문을 현재작업등록부로 설정하는것이다. Basename지령은 경로의 마지막입력을 제외한 모든것을 지운다. 재촉문은 현재등록부를 보여 준다.
9. trap지령은 신호처리지령이다. 쉘에서 탈퇴할 때 즉 등록탈퇴할 때 .logout 파일이 실행된다.
10. clear지령은 현시장치을 지운다.

재촉문 쉘은 대화형으로 사용될 때 입력을 재촉한다. 재촉문이 나타나면 지령입력을 시작할수 있다. Bourne쉘은 2개의 재촉문 즉 1차재촉문인 화페기호($)와 2차재촉문인 크기부호(>)를 제공한다. 재촉문들은 쉘이 대화형으로 실행될 때 현시된다. 사용자들은 이 재촉문들을 변경시킬수 있다. 변수 PS1은 초기에 화페기호($)로 설정되는 1차재촉문이다. 재촉문은 등록가입할 때 나타나는데 이때 쉘은 지령들을 건반으로 입력하기를 대기한다. 변수 PS2는 2차재촉문인데 초기에 크기부호로 설정된다.

이것은 지령을 부분적으로 입력한 다음 행바꾸기건을 누를 때 나타난다. 1차, 2차재촉문들은 변화시킬수 있다.

1차재촉문 화페기호는 기정의 1차재촉문이다. 재촉문은 변경시킬수 있다. 재촉문은 보통 사용자초기화파일 .profile에서 정의된다.

실례 8-3

1. $ **ps1="'uname-n>'"**
2. *chargers>*

설명

1. 기정1차재촉문은 화페기호($)이다. PS1재촉문은 콤퓨터이름(uname-n)과 기호 >로 재설정된다(거꿀인용부호와 단일인용부호를 혼돈하지 말아야 한다.).
2. 새로운 재촉문이 현시된다.

2차재촉문 PS2재촉문은 2차재촉문이다. 그 값은 표준오유에 현시되는데 표준오유는 기정으로 현시장치이다. 이 재촉문은 하나의 지령을 완성하지 않고 행복귀를 눌렀을 때 나타난다.

실례 8-4

```
1. $ echo "Hello
2. > there"
3. Hello
   there
4. $
5. $ PS2="- - - ->"
6. $ echo Hi
7. - - - - - ->
   - - - - - ->
   - - - - - -> there'
   Hi

   there
   $
```

설명

1. 2중인용부호는 문자렬 "Hello"뒤에 놓여야 한다.
2. 행바꾸기가 입력될 때 2차재촉문이 나타난다. 닫는 2중인용부호가 입력되기전까지 2차재촉문이 현시된다.
3. echo지령의 결과가 현시된다.
4. 재촉문이 현시된다.
5. 2차재촉문이 재설정된다.
6. 단일인용부호가 문자렬 Hi뒤에 놓여야 한다.
7. 행바꾸기가 입력될 때 새로운 2차재촉문이 나타난다. 닫는 단일인용부호가 입력될 때까지 2차재촉문이 현시된다.

탐색경로 Bourne쉘은 path변수를 사용하여 지령행에 입력된 지령들을 찾는다. 경로는 지령들을 탐색할 때 쉘이 사용하는 옹근두점에 의해 분리되는 등록부목록이다. 탐색은 왼쪽에서 오른쪽으로 진행된다. 경로의 끝에 있는 점은 현재작업등록부를 현시한

다. 만일 지령이 경로에 있는 임의의 등록부에서도 발견되지 않으면 Bourne쉘은 표준오유에 통보 filename:notfound를 보낸다. 경로는 .profile파일에서 설정되여야 한다. 만일 경로에 점이 포함되여 있지 않을 때 현재작업등록부로부터 지령이나 스크립트를 실행하려면 스크립트이름앞에 ./program _name과 같이 ./을 붙여서 쉘이 그 프로그람을 찾을수 있도록 해야 한다.

실례 8-5

(Printing the PATH)

1. $ **echo $PATH**
   /home/gsal2/bin:/usr/ucb:/usr/bin:/usr/local/bin:/usr/bin:/usr/local/bin:.

(Setting the PATH)

2. $ **PATH=$HOME:/ttsr/ucb:/usr:/usr/bin:/usr/local/bin:.**
3. $ **export PATH**

설명

1. $PATH를 출력하여 PATH변수의 값을 현시한다. 경로는 옹근두점으로 분리되는 요소들의 목록으로 이루어 지는데 왼쪽에서 오른쪽으로 탐색된다. 경로의 끝에 있는 점은 사용자의 현재작업등록부를 나타낸다.
2. 경로를 설정하기 위하여 PATH변수에 옹근두점으로 분리된 등록부목록을 대입한다.
3. 경로를 넘겨 자식프로쎄스들이 경로에 접근할수 있게 한다.

hash지령 hash지령은 지령을 탐색하는 효률을 높이기 위하여 쉘이 사용하는 내부하쉬표를 조종한다.

쉘은 지령이 입력될 때마다 경로를 탐색하지 않고 처음에 지령을 입력할 때 탐색경로를 사용하여 그 지령을 찾은 다음 그것을 쉘기억기에 있는 표에 기억시킨다. 다음번에 같은 지령을 사용하는 경우 쉘은 하쉬표를 사용하여 그것을 찾는다. 이렇게 하면 완전한 경로를 탐색하는것보다 더 빨리 지령을 호출할수 있다. 만일 어떤 지령을 자주 사용하는 경우 그것을 하쉬표에 추가할수 있다. 하쉬표로부터 지령들을 없앨수도 있다. hash지령의 출력은 쉘이 표를 사용하여 지령을 찾은 회수(성공한 회수)와 그 지령을 찾는 상대적비용 즉 지령을 찾기전에 쉘이 탐색을 얼마나 진행했는가를 나타낸다. -r추가선택을 가지는 hash지령은 하쉬표를 지운다.

실례 8-6

1. $ **hash**
   hits cost command
   3 8. /usr/bin/date
   1 8. /usr/bin/who
   1 8. /usr/bin/ls
2. $ **hash vi**
   3 8. /usr/bin/date
   1 8. /usr/bin/who
   1 8. /usr/bin/ls

0 6 /usr/ucb/vi

3. $ **hash –r**

설명

1. hash지령은 현재내부하쉬표에 기억된 지령들을 현시한다. 목록화된 지령들을 지령행에 입력할 때 쉘은 그것들을 찾기 위하여 탐색경로를 탐색하지 않아도 된다. 이렇게 하면 시간을 절약한다. 그렇지 않으면 쉘은 디스크로 가서 경로를 탐색해야 한다. 새로운 지령을 입력할 때 쉘은 먼저 경로를 탐색하고 그것을 하쉬표에 넣는다. 다음번에 그 지령을 사용할 때 쉘은 그것을 기억기에서 찾는다.
2. hash지령은 인수들을 가질수 있다. 보호하려는 지령들의 이름들은 미리 하쉬표에 기억된다.
3. –r추가선택들을 가지는 hash지령은 하쉬표를 지운다.

dot지령 dot지령은 Bourne쉘의 내부지령이다. 그것은 스크립트이름을 인수로 취한다. 스크립트는 현재쉘의 환경에서 실행된다. 즉 자식프로쎄스가 파생되지 않는다.

스크립트에 있는 모든 변수모임은 현재쉘환경의 부분으로 된다. 마찬가지로 현재쉘에 있는 모든 변수모임은 스크립트환경의 부분으로 된다. dot지령은 보통 .profile파일이 변경되면 그것을 다시 실행시키는데 사용된다. 실례로 EDITOR나 TERM변수와 같이 설정된것들중에서 어느하나가 등록가입후에 변화되였으면 탈퇴하였다가 다시 등록가입하지 않고도 dot지령을 사용하여 .profile을 재실행시킬수 있다.

실례 8-7

$. **.profile**

설명

dot지령은 쉘에 있는 초기화파일 .profile을 실행시킨다.

국부변수와 전역변수는 이 쉘에서 다시 정의된다. dot지령을 사용하면 등록탈퇴하였다가 다시 등록가입할 필요가 없다.[1]

2. 지령행

쉘은 등록가입한후 기정으로 1차재촉문인 $를 현시한다. 쉘은 지령을 해석하는 해석프로그람이다. 쉘이 대화형으로 실행될 때 그것은 말단에서 지령을 읽어 들이고 지령행을 단어들로 나눈다.

한개의 지령행은 공백에 의해 구분되는 한개이상의 단어들로 이루어 지는데 Enter건을 누를 때 생기는 행바꾸기로 끝난다.

첫번째 단어는 지령이고 그다음 단어들은 지령의 인수들이다. 지령은 ls나 pwd와 같이 UNIX실행형프로그람일수도 있고 cd나 test같이 내부지령일수도 있으며 쉘스크립트일수도 있다. 이 지령에는 쉘이 지령행을 구문해석할 때 해석해야 할 메타문자라고 하는

---

1. profile이 한개 스크립트로 직접 실행되면 보조쉘이 파생된다. 이때 변수들은 보조쉘에서는 설정되지만 등록가입쉘(어미쉘)에서는 설정되지 않는다.

특수문자들이 있을수 있다. 만일 지령행이 길어서 다음행에 계속 입력하려면 먼저 거꿀빗선 \ 를 입력하고 행을 바꾸어야 한다. 이때 그 지령행이 끝날 때까지 2차재촉문이 나타난다.

탈퇴상태 한개 지령이나 프로그람은 끝나면 탈퇴상태를 어미프로쎄스에 넘긴다. 탈퇴상태는 0과 255사이의 수이다. 습관상 한 프로그람이 탈퇴할 때 넘겨 진 상태가 0이면 그 지령은 실행에서 성공한것으로 본다. 만일 탈퇴상태가 0이 아니면 그 지령은 실패한것이다. 쉘상태변수 ?는 실행된 마지막지령의 탈퇴상태값으로 설정된다. 프로그람이 성공인가 실패인가는 그 프로그람을 작성한 작성자에게 달려 있다.

실례 8-8

1. $ grep “John” /etc/passwd
   john:MgVyBsZJavd16s:9496:40:John Doe:/home/falcon/john:/bin/sh
2. $ **echo $?**
   0
3. $ grep “nicky” /etc/passwd
4. $ **echo $?**
   1
5. $ grep “scott” /etc/passsswd
   grep: /etc/passsswd.: No such file or directory
6. $ **echo $?**
   2

설명

1. grep프로그람은 /etc/passwd파일에서 패턴john을 탐색하여 성공한다. /etc/passwd로부터 행이 현시된다.
2. 변수 ?는 grep지령의 탈퇴값으로 설정된다. 령은 성공을 의미한다.
3. grep프로그람은 /etc/passwd파일에서 사용자 nicky를 찾을수 없다.
4. grep프로그람이 패턴을 찾지 못하면 탈퇴상태를 1로 되돌린다.
5. /etc/passsswd파일이 열릴수 없기때문에 grep는 실패한다.
6. grep는 파일을 찾을수 없으면 탈퇴상태를 2로 되돌린다.

지령행에서 다중지령 한개의 지령행은 다중지령들로 이루어 질수 있다. 매개 지령은 반두점에 의해 구별되고 행바꾸기로 끝난다.

실례 8-9

$ **ls; pwd; date**

설명

지령들은 새행이 시작될 때까지 왼쪽에서 오른쪽으로 하나하나 실행된다.

지령의 그룹화 지령들을 묶어서 모든 출력이 다른 지령에 파이프되거나 파일로 방향바꾸기되게 할수 있다.

실례 8-10

```
$ (ls ; pwd ; date) > outputfile
```

설명

매 지령의 출력은 outputfile이라고 하는 파일에 보내진다.
괄호안에 공백이 있어야 한다.

지령의 조건실행 조건실행을 하는 2개의 지령문자렬들은 특수메타문자 &&와 ‖로 분리된다. 이 메타문자의 오른쪽에 있는 지령은 왼쪽에 있는 지령의 탈퇴조건에 따라 실행될수도 있고 실행되지 않을수도 있다.

실례 8-11

```
$ cc prgm1.c -o prgml && prgml
```

설명

첫번째 지령이 성공하면(탈퇴상태가 령이면) 뒤에 있는 지령이 실행된다. 즉 cc 프로그람이 prgm1.c를 번역할수 없으면 실행형프로그람인 prgm1이 실행된다.

실례 8-12

```
$ cc prog.c>& err  ||  mail bob < err
```

설명

첫번째 지령이 실패하면(령이 아닌 탈퇴상태를 가지면) ‖의 뒤에 있는 지령이 실행된다. 즉 cc프로그람이 prog.c를 번역할수 없으면 err라고 하는 파일에 오유가 보내지고 사용자 bob에 err파일이 전송된다.

배경에서 지령 보통 지령을 실행할 때 그것은 전경에서 실행되고 재촉문은 그 지령이 실행을 끝낼 때까지 다시 나타나지 않는다. 그런데 이렇게 지령이 끝날 때까지 기다리는것이 언제나 편리한것은 아니다. 지령행의 끝에 &을 붙이면 쉘은 즉시에 자기 재촉문을 현시하고 동시에 배경에서 그 지령을 실행한다. 이때 다른 지령을 시작하기 위해 기다리지 않아도 된다. 배경과제의 출력은 그것이 처리될 때 현시장치에 전송된다. 따라서 배경에서 지령을 실행시키려고 하면 그 지령의 출력을 파일에 방향바꾸기하던가 인쇄기 같은 다른 장치에 파이프하여 그 출력이 지금 진행하고 있는것을 간섭하지 않게 해야한다. $! 변수의 값은 배경에 있는 마지막과제의 PID번호이다.

실례 8-13

1. **$ man sh | lp&**
2. *[1] 1557*
3. **$ kill -9 $!**

설명

1. man지령의 출력(sh지령을 위한 안내페지)은 인쇄기에 파이프된다. 지령행의 끝에 있는 &는 과제를 배경에 넣는다.
2. 현시장치에 2개의 수가 나타난다. 중괄호안에 있는 수는 이 과제가 배경에 첫번째로 놓인다는것을 현시하고 두번째 수는 PID 즉 이 과제의 프로쎄스 식별번호이다.
3. 쉘재촉문이 즉시에 나타난다. 프로그람이 배경에서 실행되는 동안 쉘은 전경에서 또 다른 지령을 대기한다.
4. !변수는 배경에 제일 마지막으로 들어 온 과제의 PID이다. 만일 그것을 제때에 얻으면 이 일감이 인쇄대기로 들어 가기전에 그것을 중지시킨다.

3. 메타문자(통용기호)

메타문자들은 자기자체가 아닌 다른것을 현시하는데 사용되는 특수문자들이다. 쉘메타문자들을 통용기호라고 부른다. 표 8-1에서 메타문자들과 그 기능을 보여 주었다.

표 8-1. 쉘메타문자

메타문자	의 미
\	다음에 오는 문자를 문자그대로 해석한다.
&	배경에서 처리한다.
;	지령들을 분리한다.
$	변수들을 대입한다.
?	한개의 단일문자를 대신한다.
[abc]	문자들의 모임으로부터 한문자를 대신한다.
[!abc]	문자들의 모임밖의 한문자를 대신한다.
*	임의의 수의 문자들을 대신한다.
(cmds)	보조쉘에서 지령들을 실행한다.
{cmds}	현재쉘에서 지령들을 실행한다.

4. 파일이름바꿔넣기

쉘은 지령행을 평가할 때 파일이름이나 경로이름을 간략화하기 위해 어떤 문자들의 모임과 맞먹는 메타문자들을 사용한다. 표 8-2에서 보여 준 파일이름바꿔넣기메타문자들은 자모로 현시된 파일이름들의 모임으로 확장된다. 메타문자를 파일이름들로 넘기는 과정을 파일이름바꿔넣기라고 한다. 메타문자를 사용하였을 때 그에 대응하는 파일이름이 없으면 쉘은 메타문자를 그대로 한개의 문자로 처리한다.

표 8-2. 쉘메타문자와 파일이름대입

메타문자	의 미
*	임의의 수의 문자들을 대신한다.
?	한개 문자만 대신한다.

(표계속)

메타문자	의 미
[abc]	모임에서 한문자 a, b 또는 c를 대신한다.
[a-z]	a부터 z 사이의 한문자를 대신한다.
[!a-z]	a부터 z 사이의 범위밖에 있는 한문자를 대신한다.
\	메타문자로 쓰이지 않게 한다.

별표 별표는 파일이름에 있는 임의의 수의 문자들을 대신하는 통용기호이다.

실례 8-14

1. **$ ls ***
   abc abcl abcl22 abcl23 abc2 filel fllel.bak file2 file2.bak none nonsense nobody nothing nowhere one
2. $ **ls *.bak**
   filel.bak file2.bak
3. $ **echo a***
   ab abcl abcl22 abcl23 abc2

설명

1. *는 현재작업등록부에 있는 모든 파일들로 확장된다. 모든 파일들은 인수로서 is에 대입되여 현시된다.
2. 임의의 수의 문자들로 시작되고 .bak로 끝나는 모든 파일들이 현시된다.
3. a로 시작되고 그다음에 임의의 수의 문자들이 오는 모든 파일들이 일치되여 echo지령에 인수로서 넘겨 진다.

물음표 물음표는 파일이름에서 한개 문자를 대신한다. 파일이름에 한개이상의 물음표가 있을 때 쉘은 물음표를 파일이름에서 그와 맞먹는 문자로 바꾸어 파일이름바꿔넣기를 진행한다.

실례 8-15

1. $ **ls**
2. *abc abc122 abc2 file1.bak file2.bak nonsense nothing one*
   abc123 file1 file2 none noone nowhere
3. $ **ls a?c?**
   abc1 abc2
4. $ **ls ??**
   ?? *not found*
5. $ **echo abc???**

abc122 abc123

6. $ **echo ??**
   ??

설명

1. 현재등록부에 있는 파일들이 현시된다.
2. a로 시작되여 한개 문자 c, 그다음에 한개 문자가 더 오는 파일이름들이 일치되여 현시된다.
3. 2개 문자로만 된 파일이름이 있으면 현시된다. 2개 문자파일이 없으므로 물음표는 문자그대로 파일이름으로 취급된다.
4. abc로 시작해서 3개문자가 그뒤에 더 놓이는 파일이름들이 확장되여 echo 지령에 의해 현시된다.
5. 등록부에는 2개 문자로만 된 파일이 없다. 쉘은 물음표에 대응한 문자를 찾지 못하면 그것을 문자그대로 물음표로 취급한다.

중괄호 이 괄호는 문자들의 모임이나 렬에 있는 한개 문자를 포함하고 있는 파일이름들을 일치시키는데 사용된다.

실례 8-16

1. $ **ls**
   abc abc122 abc2 file1.bak file2.bak nonsense nothing
   one abc1 abc123 file1 file2 none noone nowhere
2. $ **ls abc[123]**
   abc1 abc2
3. $ **ls abc[l-3]**
   abc1 abc2
4. $ **ls [a-z] [a-z] [a-z]**
   abc1 abc2
5. $ **ls [!f-z] ???**
   abc1 abc2
6. $ **ls abc12[23]**

   abc122 abcl23

설명

1. 현재작업등록부에 있는 모든 파일들이 현시된다.
2. 파일이름이 abc로 시작되여 그뒤에 1, 2, 3중에서 어느 하나가 오는 파일이름이 있으면 이러한 4개 문자로 된 모든 파일이름들이 현시된다. 괄호안에 있는 문자모임에서 한개의 문자만이 일치된다.
3. abc로 시작되여 1부터 3사이의 한개 수자가 놓이는 4개 문자로 된 파일이름이 있으면 현시된다.
4. 3개 문자로 된 파일이름들중에서 3개의 자모소문자들로만 이루어 진 파일이

름이 있으면 현시된다.
5. 4개 문자로 된 파일이름들중에서 첫번째 문자가 f와 z사이의 소문자가 아니고 그뒤에 임의의 3개 문자가 오는 파일이름들이 현시된다(실례로 ???).
6. 파일이름이 abc122 또는 abc123이면 현시된다.

메타문자에서 벗어나기 메타문자를 문자그대로 사용하려면 거꿀빗선부호를 사용하여 메타문자가 해석되지 않게 한다.

실례 8-17

1. $ **ls**
   abc filel youx
2. $ **echo How are you?**
   How are youx
3. $ **echo How are you\ ?**
   How are you?
4. $ **echo** When does **this** line \
   > ever end\ ?
   When does this line ever end?

설명

1. 현재작업등록부에 있는 파일들이 현시된다(파일 youx를 주목하기 바란다.).
2. 쉘은 파일이름을 물음표까지 확장한다. 현재등록부에서 y-o-u로 시작하여 그다음에 한개 문자가 오는 임의의 파일이 선택되여 문자렬에서 대입된다. 파일이름 youx는 문자렬에서 대입되여 문자렬이 바라던것과는 달리 How are youx로 된다.
3. 거꿀빗선부호를 물음표앞에 붙여서 쉘이 물음표를 통용기호로 해석하지 않게 한다.
4. 행바꾸기앞에 거꿀빗선을 추가함으로써 그것이 메타문자로 사용되지 않게 한다. 2차재촉문은 문자렬이 행바꾸기로 끝날 때까지 현시된다. 물음표(?)가 메타문자에서 벗어 나 파일이름완성에 사용되지 않는다.

5. 변수

두가지 형태의 변수 즉 국부변수와 환경변수가 있다. 일부 변수는 사용자에 의해서 작성되는것이고 다른것들은 특수한 쉘변수들이다.

국부변수 국부변수는 정의된 쉘에게만 알려 지는 주어 진 값이다. 변수이름은 반드시 자모문자나 밑선으로부터 시작해야 한다. 나머지 문자들은 자모문자 0부터 9까지의 수자 그리고 밑선이 될수 있다. 변수이름의 끝은 임의의 문자가 될수 있다. 변수의 값을 대입할 때 같기부호주위에 공백을 두지 않아도 된다. 변수에 빈값을 설정하기 위해서는 같기부호다음에 행바꾸기를 놓으면 된다.[2] 화페기호를 변수앞에 붙여서 그 변수에 기억된 값을 꺼낼수 있다.

---

2. 어떤 값이나 빈값으로 설정된 변수는 set 지령에 의해 현시되고 이와 반면에 설정되지 않은 변수는 현시되지 않는다.

국부변수설정

실례 8-18

1. $ **round=world**
   $ echo $round
   world
2. $ **name="Peter Piper"**
   $ **echo $name**
   Peter Piper
3. $ **x=**
   $ **echo $x**
4. $ **file.bak="$HOME/junk"**
   file.bak□/home/jody/ellie/junk: not found

설명

1. 변수 round의 값은 world로 대입된다. 쉘은 변수이름앞에 화폐기호가 있으면 변수바꿔넣기를 한다. 변수의 값이 현시된다.
2. 변수 name의 값은 "Peter Piper"로 된다. 인용부호는 쉘이 지령행을 해석할 때 문자렬을 개개의 단어로 나누지 않도록 공백을 숨기기 위해 필요하다. 변수의 값이 현시된다.
3. 변수 x에는 값이 대입되지 않는다. 그것은 빈값으로 된다. 빈문자렬인 빈값이 현시된다.
4. 변수이름에 있는 점이 규칙에 맞지 않는다. 변수이름으로 될수 있는 문자들은 수자, 문자 그리고 밑선이다. 쉘은 이 문자렬을 한개 지령으로 실행하려고 한다.

국부변수의 유효범위 국부변수는 그것이 작성된 쉘에서만 유효하다. 그것은 보조쉘들에 넘겨 지지 않는다. $$변수는 현재쉘의 PID를 현시하는 특수변수이다.

실례 8-19

1. $ **echo $$**
   1313
2. $ **round=world**
   $ **echo $round**
   world
3. $ **sh** *# Start a subshell*
4. $ **echo $$**
   1326
5. $ **echo $round**
6. $ **exit** *# Exits this shell, return to parent shell*
7. $ **echo $$**
   1313

8. $ **echo $round**
   world

설명

1. $$변수의 값은 현재쉘의 PID 를 현시한다. 이 쉘의 PID는 1313이다.
2. 국부변수 round의 값은 문자렬값 world이다. 이 변수의 값이 현시된다.
3. 새로운 Bourne쉘이 시작된다. 이것을 보조쉘 또는 자식쉘이라고 한다.
4. 이 쉘의 PID는 1326이다. 어미쉘의 PID는 1313이다.
5. 변수 round는 이 쉘에서 정의되지 않는다. 빈행이 인쇄된다.
6. exit지령은 이 쉘을 끝내고 어미쉘로 되돌아 간다(Control-D도 쉘을 끝낸다.).
7. 어미쉘로 되돌아 간다. 그의 PID가 현시된다.
8. 변수 round의 값이 현시된다.

읽기전용함수설정 읽기전용함수는 다시 정의할수 없고 설정을 해제할수도 없다.

실례 8-20

1. $ **name=TOM**
2. $ **readonly name**
   $ **echo $name**
   Tom
3. $ **unset name**
   name: readonly
4. $ **name = Joe**
   name: readonly

설명

1. 국부변수 name의 값은 Tom으로 된다.
2. 변수가 읽기만 하도록 설정된다.
3. 읽기전용함수는 설정을 해제할수 없다.
4. 읽기전용함수는 다시 정의할수 없다.

환경변수 환경변수는 그것이 작성된 쉘과 그 쉘로부터 파생된 임의의 보조쉘이나 프로쎄스들에서 모두 사용할수 있다. 습관상 환경변수는 대문자로 쓴다. 환경변수는 넘겨지는 변수이다.

변수가 작성된 쉘을 어미쉘이라고 한다. 만일 새로운 쉘이 그 어미쉘에서부터 시작되면 그것을 자식쉘이라고 한다. HOME, LOGNAME, PATH, SNELL과 같은 환경변수들은 /bin/login프로그람에 의해서 등록가입하기전에 설정된다. 환경변수들은 보통 사용자의 홈등록부에 있는 .profile파일에서 정의되고 기억된다. 표 8-3에 환경변수들의 목록을 보여 주었다.

표 8-3. Bourne쉘의 환경변수

환경변수	값
PATH	지령에 대한 탐색경로

(표계속)

환경변수	값
HOME	홈등록부인데 규정된 등록부가 없을 때 cd에 의해 사용된다.
IPS	내부마당분리기이다. 보통 공백, 타브, 행바꾸기이다.
LOGNAME	사용자의 등록가입이름
MAIL	이 파라메터가 우편파일의 이름으로 설정되고 파라메터가 설정되지 않으면 쉘은 사용자에게 규정된 파일에 우편이 도착했다는것을 알린다.
MAIL CHECK	이 파라메터는 쉘이 MAILPATH나 MAIlL파라메터에 의해 지정된 파일에 우편이 도착했는가를 몇번(초)검사하는가를 지정한다. 기정값은 600초(10분)이다. 만일 0으로 설정되면 쉘은 매 초기재촉문을 설정하기전에 검사한다.
MAILPATH	웅근두점으로 분리되는 파일이름들의 목록이다. 이 파라메터가 설정되면 쉘은 사용자에게 임의의 지정된 파일에 우편이 도착했다는 것을 알려 준다. 매 파일이름뒤에는 수정시간이 변화될 때 인쇄되는 표식 %와 통보가 올수 있다. 기정통보는 You have mail이다.
PWD	현재작업등록부이다.
PS1	1차재촉문문자렬이다. 기정값은 화페기호이다.
PS2	2차재촉문문자렬이다. 기정값은 크기부호 >이다.
SHELL	쉘은 호출될 때 이 이름에 대한 환경을 조사한다. 쉘은 PATH, PS1, PS2, MAIL CHECK와 IFS 에 기정값을 준다. HOME과 MAIL은 login에 의해 설정된다.

환경변수설정 환경변수를 설정하기 위해서 값을 대입한 다음이나 변수가 설정될 때 export지령을 사용한다(변수를 넘길 때 화페기호를 사용하지 말아야 한다.).

실례 8-21

```
1.  $ TERM =wyse
    $ export TERM
2.  $ NAME= "John Smith"
    $ export NAME
    $ echo $NAME
    John Smith
3.  $ echo $$
    319                 # pid number for pqrent shell
4.  $ sh
                        # Start a subshell
5.  $ echo $$
    340                 # pid number for new shell
6.  $ echo $NAME
    John Smith
```

7. $ **NAME="April Jenner"**
   $ **export NAME**
   $ **echo $NAME**
   April Jenner
8. $ **exit** *# Exit the subshell and go back to parent shell*
9. $ **echo $$**
   319 *# Pid number for parent shell*
10. $ **echo $NAME**

John Smith

설명

1. TERM변수의 값은 wyse로 된다. 변수가 넘겨 진다. 이때 쉘에서 시작되는 프로쎄스들은 이 변수를 계승한다.
2. 변수가 그 쉘에서 시작된 보조쉘에서도 사용할수 있도록 정의되고 넘겨 진다.
3. 이 쉘의 PID값이 인쇄된다.
4. 새로운 Bourne쉘이 시작된다. 새로운 쉘을 자식이라고 한다. 초기쉘은 이것의 어미이다.
5. 새로운 Bourne쉘의 PID는 변수 $$에 기억되고 그 값이 출력된다.
6. 어미쉘에서 설정된 변수는 이 새로운 쉘에 넘겨 져서 현시된다.
7. 변수는 April Jenner로 재설정된다. 이것은 모든 보조쉘들에 넘겨 지지만 어미쉘에는 영향을 주지 않는다. 넘겨 진 값들은 어미쉘에 넘겨 지지 않는다.
8. 자식쉘 Bourne가 탈퇴된다.
9. 어미의 PID가 다시 현시된다.
10. 변수 NAME은 자기의 원래값을 가진다. 변수들은 어미쉘에서 자식쉘에 넘겨 질 때 자기들의 값을 보존한다. 자식쉘은 어미쉘의 변수값을 변화시킬수 없다.

설정된 변수들을 현시 변수들의 값을 현시하는 2개의 내부지령 set와 env가 있다. set지령은 모든 변수들 즉 국부변수와 전역변수를 다 현시한다. env지령은 전역변수들만 현시한다.

실례 8-22

1. $ **env** *(Partial list)*
   LOGNAME ellie
   TEKMCAP sun-cmd
   USER ellie
   DISPLAY :0.0
   SHELL /bin/sh
   HOME /home/jody/ellie
   TERM sun-cmd
   LD_LIBRARY_PATH /usr/local/OW3/lib PWD /home/jody/el11e/peri

2. $ **set**
   DISPLAY :0.0
   FMHOME /usr/local/Frame-2.1X

FONTPATH /usr/local/OW3/lib/fonts
HELPPATH /usr/local/OW3/lib/locale/:/usr/local/OW3/lib/help
HOME /home/jody/ellie
HZ 100
IFS
LANGC
LD_LIBRARY_PATH /usr/local/OW3/lib
LOGNAME ellie
MAILCHECK 600
MANPATH /usr/local/OW3/share/man:/usr/local/OW3/man:/
usr/local/man:/usr/local/doctools/man:/usr/man
OPTIND 1
PATH /home/jody/ellie:/usr/local/OW3/bin:/usr/ucb:/usr/local/
doctools/bin:/usr/bin:/usr/local:/usr/etc:/etc:/usr/spool/
news/bin:/home/jody/el1ie/bin:/usr/lo
PS1 $
PS2 >
PWD /home/jody/ellie/kshprog/joke
SHELL=/bin/sh
TEKM sun-cmd
TERMCAP sun-cmd:te \ E[>4h:ti \ E[>4l:tc sun:
USER ellie
Name Tom
Places "San Francisco"

설명

1. env지령은 모든 환경(넘겨 진)변수들을 현시한다. 이 변수들은 습관상 대문자로 쓴다. 이 변수들은 그것들이 정의된 프로쎄스들로부터 임의의 자식 프로쎄스들에 넘겨 진다.
2. set지령은 추가선택이 없을 때 모든 설정변수들 즉 국부변수와 넘겨 진 변수(빈값으로 설정된 변수들도 포함)들을 다 인쇄한다.

변수설정해제 국부변수나 환경변수들은 다 읽기전용변수로 설정되지 않으면 unset지령을 사용하여 설정을 해제할수 있다.

실례 8-23

unset name; unset TERM

설명

unset지령은 쉘기억기에서 변수를 지운다.

변수값인쇄: echo지령 echo지령은 자기의 인수들을 표준출력장치로 출력하며 Bourne쉘과 C쉘에서 주로 사용된다. Korn쉘은 내부인쇄지령을 가지고 있다. echo지령

에는 여러가지가 있다. 실례로 버클리(BSD)의 echo는 System V의 echo와 다르다. 충분한 경로이름을 지정하지 않으면 echo지령의 내부지령판본을 사용한다. 내부지령판본은 사용하는 UNIX의 판본을 반영한다. System V의 echo는 많은 조종문자들을 사용하게 하지만 BSD판본은 그렇게 하지 못한다.

표 8-4는 BSD의 echo추가선택과 조종문자들을 보여 준다.

표 8-4. BSD의 echo추가선택과 System V의 탈퇴조종문자

추가선택	의 미
BSD:	
-n	출력행의 끝에서 행바꾸기를 지우기한다.
System V:	
\ b	앞의것을 지운다.
\ c	행바꾸기가 없이 행을 인쇄한다.
\ f	페지바꾸기
\ n	행바꾸기
\ r	복귀
\ t	타브
\ v	수직타브
\ \	거꿀빗선

실례 8-24

1. $ **echo The username is $LOGNAME**.
   The username is ellie.

2. $ **echo "\ t\ tHello there\ c"** # *System V*
   Hello There$

3. $ **echo -n "Hello there"** # *BSD*
   Hello there$

설명

1. echo지령은 자기의 인수들을 현시장치에 인쇄한다. 변수바꿔넣기는 echo지령이 실행되기전에 쉘에 의해 진행된다.
2. System V판본의 echo지령은 C프로그람과 류사한 탈퇴조종문자들을 지원한다. $는 쉘재촉문이다.
3. echo지령에 대한 -n추가선택은 BSD판본의 echo지령이 사용된다는것을 나타낸다. 행바꾸기가 없이 행이 인쇄된다. 탈퇴조종문자들은 이 판본의 echo에 의해서는 지원되지 않는다.

변수확장변경자 변수들은 특수한 변경자를 사용하여 검사하거나 수정할수 있다. 변경

자는 변수가 설정되여 있는가를 검사하기 위하여 지름조건검사를 할수 있게 하며 그 검사의 결과에 따라 변수에 값을 대입한다. 표 8-5에서 변수변경자들의 목록을 보여 주었다.

표 8-5. 변수변경자

변경자	값
${variable:-word}	변수가 설정되고 빈값이 아니면 그 값을 대입한다. 그렇지 않으면 word를 대입한다.
${variable:=word}	변수가 설정되거나 빈값이 아니면 그 값을 대입한다. 그렇지 않으면 word로 설정한다. 변수값은 항구적으로 대입된다. 위치파라메터는 이런식으로 대입할수 없다.
${variable:+word}	파라메터가 설정되고 빈값이 아니면 word를 대입한다. 그렇지 않으면 아무것도 대입하지 않는다.
${variable:?word}	변수가 설정되고 빈값이 아니면 그 값을 대입한다. 그렇지 않으면 word를 인쇄하고 쉘에서 벗어 난다. word가 생략되면 통보 parameter null or not set 가 인쇄된다.

변경자(-, =, +, ?)들중에서 어느 하나를 웅근두점과 함께 사용하여 변수가 설정되지 않았는가 또는 빈값인가를 검사한다. 웅근두점이 없을 때 빈값으로 설정된 변수는 설정된것으로 본다.

실례 8-25

(Assigning Temporary Default Values)

```
1.  $ fruit=peach
2.  $ echo ${fruit:-plum)
    peach
3.  $ echo ${newfruit:-apple}
    apple
4.  $ echo $newfruit
5.  $ echo $EDITOR            # More realistic example
6.  $ echo ${EDITOR:-bin/vi}
    /bin/vi
7.  $ echo $EDITOR
8.  $ name=
    $ echo ${narne-Joe}
9.  $ echo ${name:-Joe}
    Joe
```

설명

1. 변수 fruit의 값은 peach로 된다.
2. 변수 fruit가 설정되였는가를 특수변경자가 검사해 본다. 설정되였으면 그 값이 인쇄된다. 설정되지 않았으면 fruit에 plum이 대입되여 인쇄된다.
3. 변수 newfruit는 설정되지 않았다. 변수 apple이 newfruit에 림시로 대입된다.

4. 설정은 다만 림시적이다. 변수 newfruit는 설정되지 않는다.
5. 환경변수 EDlTOR는 설정되지 않았다.
6. :-변경자는 EDlTOR에 /bin/vi 를 대입한다.
7. EDlTOR는 전혀 설정되지 않았다. 아무것도 인쇄되지 않는다.
8. 변수 name은 빈값으로 설정된다. 변경자앞에 옹근두점이 없으므로 변수가 빈값으로 설정된다. 새로운 값 Joe는 name에 대입되지 않는다.
9. 옹근두점은 변수가 설정되지 않았는가 또는 빈값으로 설정되였는가를 변경자가 검사하도록 한다. 어느 경우에나 값 Joe는 name에 대입된다.

실례 8-26

(Assigning Permanent Default Values)

1. $ **name=**
2. $ **echo ${name!=Patty}**
   Patty
3. $ **echo $name**
   Patty
4. $ **echo ${EDITOR:=/bin/vi>**
   /bin/vi
5. $ **echo $EDITOR**
   /bin/vi

설명

1. 변수 name의 값은 null이다.
2. 특수변경자 :=는 변수 name이 설정되였는가를 검사한다. 만일 그것이 설정되였다면 그것은 변화되지 않는다. 그것이 빈값이거나 설정되지 않았다면 갈기부호의 오른쪽에 있는 값으로 대입된다. Patty는 변수가 빈값으로 설정되였으므로 name에 대입된다. 이 설정은 영구적이다.
3. 변수 name의 값은 여전히 Patty이다.
4. 변수 EDlTOR의 값은 /bin/vi로 설정된다.
5. 변수 EDlTOR의 값이 현시된다.

실례 8-27

(Assigning Temporary Alternate Value)

1. $ **foo=grapes**
2. $ **echo ${foo:+pears)**
   pears
3. $ **echo $foo**
   grapes
   $

설명

1. 변수 foo에 값 grapes가 대입된다.

2. 특수변경자 :+는 변수가 설정되였는가를 검사한다. 설정되였다면 pears가 림시로 foo에 대입된다. 설정되지 않았다면 빈값이 복귀된다.
3. 변수 foo는 자기의 초기값을 가진다.

실례 8-28

(Creating Error Messages Based On Default Values)

1. $ **echo ${namex:?"namex is undefined")**
   namex: namex is undefined
2. $ **echo ${y?}**
   y: *parameter null* or *not* set

설명

1. 변경자 ?:는 변수가 설정되였는가를 검사한다. 설정되지 않았으면 ?의 오른쪽에 있는 문자렬이 변수이름다음에 표준오유로 인쇄된다. 만일 스크립트안에서 설정되였다면 그 스크립트가 탈퇴한다.
2. ?뒤에 통보가 없다면 쉘이 표준통보를 표준오유로 내보낸다.

위치파라메터 위치파라메터라고 부르는 특수내부변수들은 보통 지령행으로부터 인수를 넘길 때 쉘스크립트들에서 사용되거나 함수에서 사용하여 여기에 넘겨 진 인수들의 값을 기억한다(표 8-6). 변수들이 지령행우에 놓인 위치에 의해 구분되므로 위치파라메터라고 한다. Bourne쉘은 9개의 위치파라메터들을 제공한다. 쉘스크립트의 이름은 $0변수에 기억된다. 위치파라메터는 설정될수 있으며 set지령에 의해 재설정될수도 있다.

표 8-6. 위치파라메터

위치파라메터	의 미
$0	현재쉘의 스크립트이름을 나타낸다.
$1-$9	위치파라메터 1부터 9까지를 표시한다.
$#	위치파라메터의 개수를 표시한다.
$*	모든 위치파라메터를 표시한다.
$@	2중인용부호안에 있을 때를 제외하고 $*와 같은 의미이다.
"$*"	"$1 $2 $3"와 같다.
"$@"	"$1" "$2" "$3"와 같다.

실례 8-29

1. ***$ set tim bill ann fred***
   $ echo $* # Prints all the positional parameters.
   # tim bill ann fred.
2. $ **echo $1** *# Print the first positional parameter.*
   tim

3. $ **echo $2 $3** *# Print the second and third*
   bill ann *# positional parameters.*
4. $ **echo $#** *# Prints the total number of*
   4 *# positional parameters.*
5. ***$ set a b c d e f g h I j k l m***
   $ **echo $10** *# Prints the first positional parameter*
   a0 *# followed by a zero.*
6. $ **echo $***
   a b c d e f g h i j k l m
7. $ **set filel file2 file3**
   ***$** echo \ $$#*
   $3
8. **$ eval echo \ $$#**
   file3

설명

1. set지령은 값을 위치파라메터에 대입한다. $*특수변수는 모든 파라메터모임을 포함한다.
2. 첫번째 파라메터값 tim이 현시된다.
3. 두번째, 세번째 파라메터값 bill과 ann이 현시된다.
4. 특수변수 $#는 현재 설정된 위치파라메터들의 개수를 표시한다.
5. set지령은 모든 위치파라메터들을 재설정한다. 초기설정이 파괴된다. 위치파라메터들의 번호는 9를 초과할수 없다. 첫번째 위치파라메터의 값과 번호 0이 인쇄된다.
6. $*표식은 번호가 9를 초과한다 해도 모든 파라메터들을 인쇄할수 있게 한다.
7. 위치파라메터들은 file1, file2 그리고 file3으로 재설정된다. 화폐기호가 특수문자로 사용되지 않는다. $#은 인수의 개수이다. echo지령은 $3을 현시한다.
8. eval지령은 지령을 실행하기전에 또 한번 지령행을 해석한다. 쉘은 첫번째로 해석할 때 \ $$#를 $3으로 대입하고 두번째로 해석할 때에는 $3의 값을 file3으로 교체한다.

기타 특수변수 쉘은 한개의 문자로 이루어 진 특수변수들을 가지고 있다. 문자앞에 있는 화폐기호는 변수에 기억된 값을 호출할수 있게 한다(표 8-7).

표 8-7. 특수변수

변수	의미
$	쉘의 PID
-	현재 설정된 sh추가선택
?	마지막으로 실행된 지령의 출력값
!	마지막으로 배경에 넣어 진 과제의 PID

실례 8-30

1. $ **echo The pid of this shell is $$**
   The pid of this shell is 4725

2. $ **echo The options for this shell are $-**
   The options for this shell are s

3. $ **grep dodo /etc/passwd**
   $ **echo $?**
   1

4. $ **sleep 25&**
   4736
   $ **echo $!**
   4736

설명

1. $변수는 이 프로쎄스에 대한 PID의 값을 가진다.
2. -변수는 대화형Bourne쉘에 대한 모든 추가선택을 현시한다.
3. grep지령은 /etc/passwd파일에서 문자렬 dodo를 탐색한다. 변수 ?는 마지막으로 실행된 지령의 출력값을 가진다. grep로부터 귀환된 값이 1이므로 grep는 탐색에서 실패한것으로 된다. 출력값이 0일 때 성공이다.
4. !변수는 배경에 마지막으로 놓인 지령의 PID번호를 가진다. sleep지령에 추가된 &는 지령을 배경으로 보낸다.

6. 인용부호찍기

특수메타문자가 해석되지 않도록 인용부호를 리용한다. 이 방법에는 세가지 즉 거꿀빗선, 단일인용부호 그리고 2중인용부호가 있다. 표 8-8에서 보여 준 문자들은 쉘에 대하여 특수한것이며 반드시 인용부호속에 넣어야 한다.

표 8-8. 인용부호를 요구하는 특수메타문자

메타문자	의 미
;	지령분리기
&	배경처리
()	지령묶음; 보조쉘을 만든다.
{}	지령묶음; 보조쉘을 만들지 않는다.
|	파이프
<	입력방향바꾸기
>	출력방향바꾸기
newline	지령끝

(표계속)

메타문자	의 미
space/tab	단어구분기
$	변수바꿔넣기문자
*[]?	파일이름확장을 위한 쉘메타문자.

단일인용부호와 2중인용부호는 반드시 닫겨 져야 한다. 단일인용부호는 $, *, ?, |, >, <와 같은 특수메타문자가 해석되지 않게 한다. 2중인용부호도 특수메타문자가 해석되지 않게 하지만 변수와 지령바꿔넣기문자(화폐기호와 거꿀인용부호)는 처리되게 한다.

단일인용부호는 2중인용부호를 보호하고 2중인용부호는 단일인용부호를 보호한다. Bourne쉘은 인용부호를 닫지 않았을 때 그것을 알려 주지 못한다. 대화형으로 실행될 때 괄호가 닫겨 지지 않으면 2차재촉문이 나타난다. 쉘스크립트에서 파일이 조사될 때 인용부호가 닫기지 않으면 쉘은 다음의 유용한 인용부호로 그것을 닫으려고 한다. 만일 쉘이 다음의 유용한 인용부호로 닫을수 없으면 프로그람은 실패하고 통보 'end of file' unexpected가 말단장치에 나타난다. 인용부호는 우수한 쉘프로그람작성자에게도 혼란을 줄수 있다. 쉘인용부호찍기규칙을 부록 3에서 참고하기 바란다.

**거꿀빗선 ** 거꿀빗선은 한개 문자를 해석되지 않게 하는데 사용된다. 거꿀빗선은 단일인용부호속에 있을 때 해석되지 않는다. 거꿀빗선은 2중인용부호안에 있을 때 화폐기호($), 거꿀인용부호(' ') 그리고 거꿀빗선자체를 해석되지 않게 한다.

실례 8-31

1. $ **echo Where are you going\ ?**
   Where are you going?
2. $ **echo Start on this line and **
   > go to the next line.
   Start on this line and go to the next line.
3. $ **echo \ **
   \
4. $ **echo '\ \ '**
   \ \
5. $ **echo '\ $5.00'**
   \ *$5.00*
6. $ **echo "\ $5.00"**
   $5. 00

설명

1. 거꿀빗선은 쉘이 물음표를 파일이름으로 치환하지 않도록 한다.
2. 거꿀빗선은 행바꾸기를 특수문자로 사용되지 않도록 하여 다음행이 이 행의 부분으로 되게 한다.

3. 거꿀빗선자체가 특수문자이므로 다음에 오는 거꿀빗선이 해석되지 않게 한다.
4. 거꿀빗선은 단일인용부호안에 있을 때 해석되지 않는다.
5. 단일인용부호안에 있는 모든 문자는 문자그대로 처리된다. 여기서 거꿀빗선은 어떤 다른 기능도 수행하지 않는다.
6. 거꿀빗선이 2중인용부호안에 있으면 그것은 화페기호가 변수바꿔넣기를 위해 해석되지 않도록 한다.

단일인용부호 단일인용부호들은 반드시 닫겨 져야 한다. 이것들은 모든 메타문자들이 해석되지 않도록 한다. 단일인용부호를 인쇄하려면 그것을 2중인용부호에 넣거나 거꿀빗선으로 보호해야 한다.

실례 8-32

1. **$ echo ‘hi there**
   > how are you?
   > When will this end?
   > When the quote is matched
   > oh’
   hi there
   how are you?
   When will this end?
   When the quote is matched
   oh
2. **$ echo ‘Don\ 't you need $5.00?’**
   Don't you need $5.00?

3. **$ echo 'Mother yelled, “Time to eat!”'**
   Mother yelled, “Time to eat ! ”

설명

1. 단일인용부호가 행에서 닫겨 지지 않았다. Bourne쉘이 2차재촉문을 현시한다. 그것은 단일인용부호가 닫기기를 기다린다.
2. 단일인용부호는 모든 메타문자들이 해석되지 않게 한다. Don’t안에 있는 웃반점은 거꿀빗선에 의해 메타문자로 해석되지 않는다. 그렇지 않으면 첫번째 괄호에 대한 닫기괄호로 되여 문자렬의 끝에 단일인용부호가 남게 된다. 이 실례에서 $와 ?는 쉘로부터 보호되며 문자 그자체로 처리된다.
3. 이 문자렬에서 단일인용부호는 2중인용부호를 보호한다.

2중인용부호 2중인용부호는 반드시 닫겨 져야 하는데 이것을 사용하여 변수와 지령바꿔넣기를 할수 있고 임의의 다른 특수메타문자들이 쉘에 의해 해석되지 않게 한다.

실례 8-33

1. **$ name=Jody**
2. **$ echo "Hi $name, I'm glad to meet you!"**
   Hi Jody, I'm glad to meet you !

3. $ **echo "Hey $name, the time is `data`"**
   Hey Jody, the time is Wed Dec 14 14:04 11 PST 1998

설명

1. 변수 name에 문자렬 Jody가 대입된다.
2. 문자렬을 둘러 싼 2중인용부호는 $name에서 $를 제외하고 모든 특수메타문자들이 해석되지 않게 한다. 변수바꿔넣기는 2중인용부호안에서 진행된다.
3. 변수바꿔넣기와 지령바꿔넣기는 2중인용부호로 닫겨 질 때 진행된다. 변수 name이 확장되고 거꿀인용부호안에 있는 date지령이 실행된다.

7. 지령바꿔넣기

지령의 출력을 변수에 대입하거나 문자렬안에서 대입할 때 지령바꿔넣기를 사용한다. 3개의 모든 쉘은 거꿀인용부호를 사용하여 지령바꿔넣기를 진행한다.[3]

실례 8-34

1. $ **name='nawk -F: '{print $1}' database'**
   $ **echo $name**
   Ebenezer Scrooge

2. $ **set 'date'**
3. $ **echo $***
   Fri Oct 19 09:35:21 PDT 2001
4. $ echo $2 $6
   Oct 2001

설명

1. nawk지령은 거꿀인용부호안에 있다. 이 지령이 실행되고 출력이 변수 name에 문자렬로 대입되여 현시된다.
2. set지령은 date지령의 출력을 위치파라메터에 대입한다. 공백은 단어들의 목록을 개개의 파라메터로 분리한다.
3. $*변수는 모든 위치파라메터들을 취한다. Date지령의 출력은 $*변수에 기억된다. 매 파라메터는 공백에 의해 분리된다.
4. 두번째와 여섯번째 파라메터가 인쇄된다.

8. 함수소개

Bourne쉘에서는 지령을 간략화하기 위한 별명기구가 없지만 함수(SVR2에서 쉘에 소개된)들을 지원한다. 지령들의 묶음을 한개 이름으로 실행하기 위하여 함수들을 사용한다. 그것들은 스크립트와 비슷한데 그것보다 더 효률적이다. 함수들은 일단 정의되면 쉘기억

3. Korn쉘은 웃방향호환성을 위해 지령바꿔넣기에 대한 거꿀인용부호를 허용하지만 그와 다른 방법도 제공한다.

기의 부분으로 되여 함수가 호출될 때 쉘은 그것을 디스크로부터 파일처럼 읽어 들이지 않아도 된다. 흔히 함수들을 사용하여 스크립트의 모듈성을 개선시킨다(이 장의 프로그람작성부분에서 론의한다.). 함수는 일단 정의되면 계속 사용할수 있다.

함수들은 보통 사용자초기화파일 .profile에서 정의된다. 함수들은 호출되기전에 반드시 정의되여야 하며 넘겨 질수 없다.

함수정의 함수이름뒤에는 한조의 빈 괄호가 놓인다. 함수정의는 반두점에 의해 구별되고 대괄호안에 놓이는 지령들의 모임으로 이루어 진다.

형식

function_name() {commands;commands;}

실례 8-35

1. $ **greet () { echo-Hello $LOGNAME,today is `data `; }**
2. $ **greet**
   Hello ellie, today is Thu Oct 4 19:56:31 PDT 2001

설명

1. 함수를 greet라고 한다.
2. greet함수가 호출될 때 대괄호안에 있는 지령들이 실행된다.

실례 8-36

1. $ **fun () { pwd; ls? date; }**
2. $ **fun**
   /home/jody/ellie/prac
   abc abcl23 filel.bak none nothing tmp
   abcl abc2 flle2 nonsense nowhere touch
   abc122 filel file2.bak noone one
   Sat Feb 24 11:15:48 PST 2001
3. $ **welcome () { echo "Hi $1 and $2"; }**
4. $ **welcome torn joe**
   Hi torn and joe
5. $ **set jane nina lizzy**
6. $ **echo $***
   jane nina lizzy
7. $ **welcome tom joe**
   hi tom and joe
8. $ **echo $1 $2**
   jane nina

설명

1. 함수 fun이 이름 지어 지고 정의된다. 이름뒤에는 대괄호안에 있는 지령들의 목록이 놓인다. 매 지령은 반두점에 의해 구별된다. 첫번째 대괄호다음에 공백을 주어야 한다. 그렇지 않으면 문법적오유로 된다. 함수는 사용되기 전에 정의하여야 한다.
2. 함수는 호출될 때 스크립트처럼 실행된다. 함수정의안에 있는 지령들이 차례로 실행된다.
3. 함수 welcome에서 2개의 위치파라메터가 사용된다. 인수가 함수에 주어 질 때 위치파라메터들이 값들로 대입된다.
4. 함수에 대한 인수 tom과 joe는 각각 $1과 $2에 대입된다. 함수에 있는 위치파라메터는 그 함수에 대해서만 사용되고 함수밖에서는 사용되지 않는다.
5. 위치파라메터들은 지령행에서 설정된다. 이 변수들은 함수에서 설정된것들과 아무 상관이 없다.
6. $*는 현재 설정된 위치파라메터들의 값을 현시한다.
7. 함수 welcome이 호출된다. 위치파라메터들에 대입된 값들은 tom과 joe이다.
8. 지령행에서 대입된 위치변수들은 함수에서 설정된것들에 의해 영향을 받지 않는다.

함수의 표시와 해제 함수와 그 정의를 표시하기 위하여 set지령을 사용한다. 함수와 그 정의는 넘겨 진 변수와 국부변수와 함께 출력된다. 함수와 그 정의는 unset지령으로 해제한다.

9. 표준입출력과 방향바꾸기

쉘은 기동할 때 3개의 파일 stdin, stdout, stderr를 계승한다.

표준입력은 보통 건반으로 입력된다. 표준출력과 표준오유는 보통 현시장치로 출력된다. 자료를 파일로부터 입력하거나 파일로 출력하려는 경우가 있을수 있다.

이때 입출력방향바꾸기를 사용할수 있다. 표 8-9는 방향바꾸기연산자들을 보여 주었다.

표 8-9. 방향바꾸기연산자

방향바꾸기연산자	기 능
<	입력을 방향바꾸기한다.
>	출력을 방향바꾸기한다.
>>	출력을 추가한다.
2>	오유를 방향바꾸기한다.
>2	오유가 출력되는 곳으로 출력을 방향바꾸기한다.
2>1	출력이 나가는 곳으로 오유를 방향바꾸기한다.

실례 8-37

```
1. $ tr '[A-Z]'  '[a-z]' < myfile
                        # Redirect input
```

2. $ **ls > lsfile** *# Redirect output*
   $ cat lsfile
   dirl
   dir2
   file1
   file2
   file3

3. $ **date >> lsfile** *# Redirect and append output*
   $ cat lsfile
   dirl
   dir2
   filel
   file2
   file3
   Mon Sept 17 12:57:22 PDT 2001

4. $ **cc prog.c 2> errfile** *Redirect error*

5. $ **find . -name \ *.c -print > foundit 2 > /dev/null**
   # Redirect output to foundit, and error to /dev null.

6. $ **find . -name \ *.c -print > foundit 2>&**
   # Redirect output and send standard error to where output is going

7. $ **echo " File needs an argument" 1>&2**
   # Send standard output to error

설명

1. 표준입력은 건반으로부터 입력되지 않고 파일 myfile로부터 UNIX의 tr지령에 방향바꾸기된다. 모든 대문자들은 소문자들로 변화된다.
2. ls지령은 출력을 현시장치에 보내지 않고 ls파일에 방향바꾸기한다.
3. data지령의 출력은 ls파일에 방향바꾸기되여 추가된다.
4. 파일 prog c가 번역된다. 번역이 실패하면 표준오유가 파일 errfile에 방향바꾸기된다. 이때 설명을 위해 오유파일을 국부guru에 보낼수 있다.
5. find지령은 현재작업등록부에서 .c로 끝나는 파일이름들을 탐색하여 파일이름들을 foundit라고 하는 파일에 인쇄한다. find지령으로부터 오유가 /dev/null에 전송된다.
6. find지령은 현재작업등록부에서 .c로 끝나는 파일이름들을 탐색하여 파일이름들을 foundit라는 파일에 인쇄한다. 오유들도 foundit에 보낸다.
7. echo지령은 자기의 통보를 표준오유에 보낸다. 그 표준출력은 표준오유와 합쳐 진다.

exec지령과 방향바꾸기 exec지령을 사용하여 새로운 프로쎄스를 시작하지 않고도 현재프로그람을 새로운것으로 바꿀수 있다. 보조쉘을 작성하지 않고 exec지령으로 표준입력과 표준출력을 변화시킬수 있다.

만일 파일이 exec에 의해 열린다면 그다음의 read지령이 파일지적자를 한번에 한행씩 파일끝까지 이동시킨다. 처음부터 다시 읽기를 시작하려면 파일을 닫아야 한다.

그러나 act와 sort와 같은 UNIX편의프로그람들을 사용하면 조작체계는 매 지령이 완성된 다음에 파일을 닫는다. 실례로 스크립트에서 exec지령을 어떻게 사용하는가를 보려면 258페지의 《순환지령》을 참고하기 바란다.

표 8-10. exec지령

exec 지령	기 능
exec ls	ls가 쉘에서 실행된다. ls가 끝나면 시작되였던 쉘은 복귀되지 않는다.
exec < filea	표준입력을 읽기 위해 filea를 연다.
exec > filex	표준출력을 쓰기 위해 filex를 연다.
exec 3< datfile	입력을 읽기 위하여 datfile을 파일서술자 3으로 연다.
sort < &3	datfile이 분류된다.
exec 4> newfile	쓰기 위하여 newfile을 파일서술자(fd) 4로 연다.
ls>&4	ls의 출력은 newfile에 방향바꾸기된다.
Exec 5<&4	fd4를 fd5에 복사한다.
esxec 3<&	fd3을 닫는다.

실례 8-38

```
1.  $ exec date
     Sun Oct 14 10:07:34 PDT 2001
         <Login prompt appears if you are in your login     shell>
2.  $ exec > temp
    $ ls
    $ pwd
    $ echo Hello
3.  $ exec > /dev/tty
4.  $ echo Hello
    Hello
```

설명

1. exec지령은 현재쉘에서 date지령을 실행한다(그것은 자식쉘을 만들지 않는다.). date지령이 현재쉘에서 실행되므로 date지령이 탈퇴할 때 쉘도 끝난다. 만일 Bourne쉘이 C쉘에서 시작되였다면 Bourne쉘은 탈퇴하고 C쉘재촉문이 나타난다. 이때 등록가입쉘에 있다면 거기서 등록탈퇴하게 된다. 쉘창문에서 대화형으로 동작하고 있다면 창문이 닫긴다.
2. exec지령은 현재쉘을 위한 표준출력을 temp파일로 연다. ls, pwd, echo로부터의 출력은 더이상 현시장치로 나가지 않고 temp로 간다.
3. exec지령은 표준출력을 말단으로 다시 연다. 이때 출력은 4행에서 보여 주는

것과 같이 현시장치으로 나간다.
4. 표준출력은 다시 말단으로 지정된다(/dev/tty).

실례 8-39

1. $ **cat doit**
   pwd
   echo hello
   date
2. $ **exec < doit**
   /home/jody/el lie/shell
   hello
   Sun Oct 14 10:07:34 PDT 2001
3. %

설명

1. doit라고 부르는 파일의 내용이 현시된다.
2. exec지령은 표준입력을 doit라고 하는 파일로 연다. 건반으로가 아니라 파일로부터 입력된다. 현재쉘에서 파일 doit로부터 오는 지령들이 실행된다. 마지막지령이 탈퇴할 때 쉘도 끝난다(그림 8-2).
3. exec지령이 완전히 끝났을 때 Bourne쉘이 탈퇴된다. C쉘재촉문이 나타난다. 이것은 어미쉘이다. exec가 끝났을 때 등록가입쉘에 있었으면 거기서 탈퇴된다. 창문에 있었으면 창문이 닫긴다.

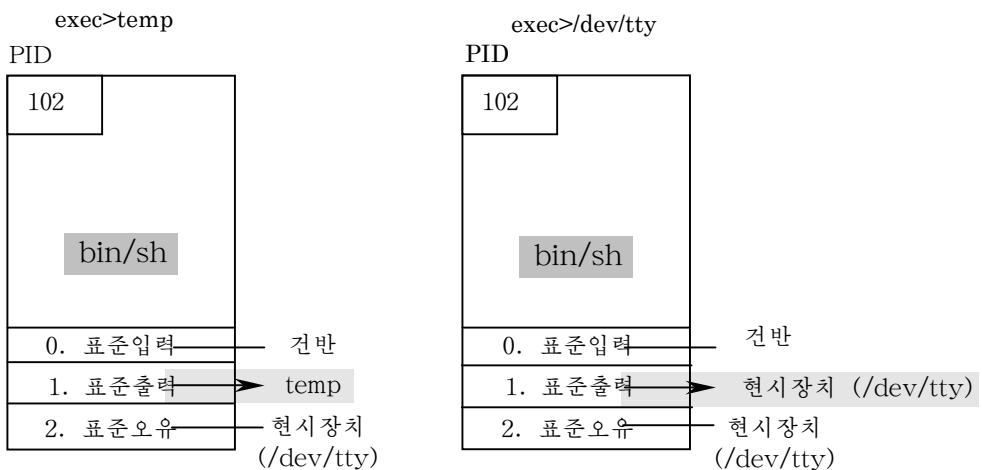



그림 8-2. exec 지령

실례 8-40

1. $ **exec 3> filex**
2. $ **who >& 3**

```
3.  $ date >& 3
4.  $ exec 3>&-
5.  $ exec 3< filex
6.  $ cat <&3
     ellie   console Oct 7     09:53
     ellie   ttyp0     Oct 7     09:54
     ellie   ttyp1     Oct 7     09:54
     ellie   ttyp2     Oct 11    15:42
     Sun Oct 14 13:31:31   PDT 2001
7.  $ exec 3<&-
8.  $ date >& 3
    Sun Oct 14 13:41:14   PDT 2001
```

설명

1. 파일서술자3(fd3)이 출력의 방향바꾸기를 위하여 filex에 대입되고 열린다(그림 8-3).
2. who지령의 출력이 파일서술자가 3인 filex에 보내진다.
3. date지령의 출력이 파일서술자 3에 보내진다. filex는 이미 열려 져 있으므로 출력이 filex에 첨부된다.
4. fd3이 닫긴다.
5. exec지령이 입력을 읽기 위해 fd3을 연다. 입력은 filex로부터 방향바꾸기된다.
6. cat프로그람이 filex에 대입된 fd3으로부터 읽는다.
7. exec지령이 fd3을 닫는다(실제로는 파일의 끝이 발견되면 조작체계가 그 파일을 닫는다.).
8. date지령의 출력을 fd3에 보내려고 할 때 파일서술자가 닫겼으므로 출력이 현시장치에 현시된다.

exec 3>filex

102

/bin/sh

0	표준입력
1	표준출력
2	표준오유
3	filex

쓰기 위해 열린다.

exec 3>filex

102

/bin/sh

0	표준입력
1	표준출력
2	표준오유
3	filex

읽기 위해 열린다.

그림 8-3. exec 와 파일서술자

10. 파이프

파이프는 파이프기호의 왼쪽에 있는 지령출력을 오른쪽에 있는 지령입력으로 보낸다. 파이프행은 한개이상의 파이프로 구성할수 있다.

다음의 3개 지령들의 목적은 등록가입한 사람들의 수(who)를 계산하고 파일(tmp)에 지령의 출력을 보존하며 wc-l을 사용하여 tmp파일에 있는 행의 개수를 계산하고(wc-l) 그다음에 tmp파일을 지우는것이다(즉 등록가입한 사람들의 수를 계산한다.). 그림 8-4와 8-5를 참고하기 바란다.

실례 8-41

1. $ **who > tmp**
2. $ **we -1 tmp**
   4 kmp
3. $ **rm tmp**

   # Using a pipe saves disk space and time.
4. $ **who | wc -1**
   4

5. ***$** du . | sort -n | sed -n '$p'*
   72388 /home/jody/ellie

설명

1. who지령의 출력은 tmp파일로 방향바꾸기된다.
2. wc-l지령은 tmp의 행수를 현시한다.
3. tmp파일이 지워 진다.
4. 파이프기능으로 3단계의 처리를 한번에 진행할수 있다. who지령의 출력은 닉명의 핵심완충기에 전달된다. wc-l지령은 완충기로부터 읽어 들이고 출력을 현시장치으로 보낸다.
5. du지령의 출력인 매 등록부당 사용되는 디스크블로크의 수는 sort지령에 파이프되여 수자적으로 분류된다. 그다음 sed지령에 파이프된다. 이 지령은 그것이 받는 출력의 마지막행을 인쇄한다.

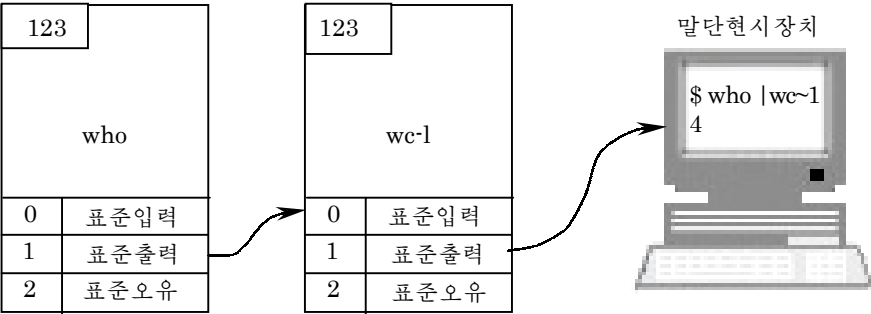



그림 8-4. 파이프

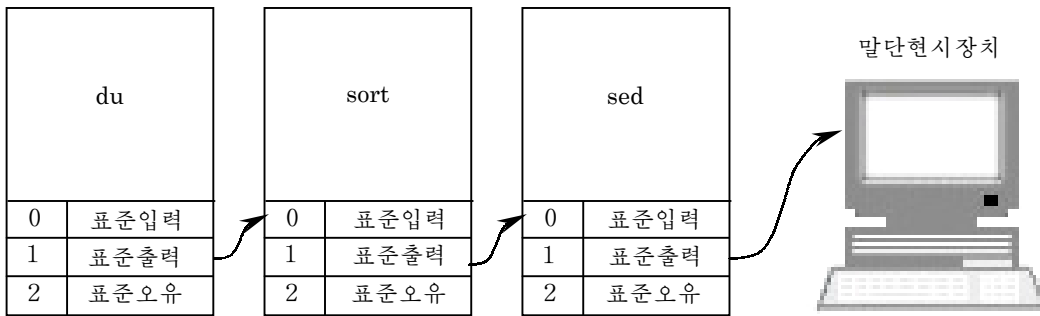



그림 8-5. 다중파이프(려파기)

11. here문서와 입력방향바꾸기

here문서는 mail, sort, cat와 같은 입력을 요구하는 프로그람을 위하여 사용자정의된 최종완료자가 나타날 때까지 직결본문을 받는다. 이것은 차림표를 만들기 위하여 쉘스크립트들에서 자주 사용된다. 입력을 받는 지령뒤에 <<기호, 사용자정의된 최종완료자나 기호 그리고 마지막에 행바꾸기가 놓인다. 본문의 다음행들은 지령에 보내지는 입력행들이다. 사용자정의최종완료자나 기호가 어떤 행의 제일 왼쪽에 혼자 나타날 때 입력이 끝난다(그 주위에 공백이 있을수 없다.). 이 단어는 Control-D를 대신하여 프로그람이 입력을 읽어 들이는것을 정지시킨다.

만일 최종완료자앞에 <<- 연산자가 놓이면 안내타브나 타브들을 최종완료자앞에 놓을수 있다. 사용자정의최종완료자나 기호는 반드시 모든 곳에서 일치되여야 한다. 다음의 실례들은 문법을 설명하기 위해 지령행에서 here문서의 사용을 설명한다. 스크립트들에서 그것들을 사용하는것이 더 실용적이다.

실례 8-42

```
1.  $ cat <<FINISH                    # FINSH is user-defined
2.  > Hello there $LOGMAME              terminator
3.  > The time is 'date'
    > I can't wait to see you !!!
4.  > FINISH                          # terminator match first
5.  Hello there ellie                  # FINISH on Line 1.
     The time is Thu Feb 8 19:42:16 PST 2001
     I can't wait to see you ! ! !
6.  $
```

설명

1. UNIX의 cat프로그람은 단어 FINISH가 행에 단독으로 나타날 때까지 입력을 접수한다.
2. 2차재촉문이 나타난다. 그다음의 본문은 cat지령에 대한 입력이다. 변수바꿔넣기는 here문서안에서 진행된다.

3. 지령바꿔넣기 'date'는 here문서내에서 진행된다.
4. 사용자정의최종완료자 FINISH가 cat프로그람에 대한 입력의 끝을 현시한다. 그것은 앞뒤에 공백을 가지지 않으며 행우에 단독으로 나타난다.
5. cat프로그람의 출력이 현시된다.
6. 쉘재촉문이 다시 나타난다.

실례 8-43

1. $ **cat <<- DONE**
   > *Hello there*
   > *What's up?*
   > *Bye now The time is 'date'.*
2. > **DONE**
   Hello there
   What's up?
   Bye now The time is Thu Feb 8 19:48:23 PST 2001.
   $

설명

1. cat프로그람은 DONE이 행우에 단독으로 나타날 때까지 입력을 접수한다. <<-연산자는 입력과 최종완료자앞에 한개이상의 타브가 놓이게 한다.
2. 최종완료자 DONE앞에 한개 타브가 놓인다. cat프로그람의 출력이 현시장치에 현시된다.

제 2 절. Bourne쉘에 의한 프로그람작성

1. 쉘스크립트작성단계

쉘스크립트는 보통 편집기에서 작성되며 설명문이 삽입된 지령들로 이루어 진다. 설명문은 앞에 표식 #이 오며 지령의 기능을 설명하는 본문으로 구성된다.

첫번째 행 스크립트의 꽉대기 왼쪽구석에 있는 첫번째 행은 스크립트행들을 실행하는 프로그람을 표시한다. 이 행은 보통 다음과 같이 쓴다.

```
#!/bin/sh
```

#!는 특수한 번호라고 부르는데 핵심부는 이것을 사용하여 스크립트에서 행들을 해석하는 프로그람을 식별한다. 이행은 스크립트의 머리부에 놓여야 한다.

설명문 설명문은 표식(#)이 앞에 놓이는 행인데 행우에 혼자 또는 스크립트지령 다음에 놓일수도 있다. 그것들은 스크립트를 설명하는데 사용된다. 때때로 설명문이 없으면 스크립트가 무엇을 하는지 리해하기가 어렵다. 비록 설명문이 중요하지만 그것들은 너무 빈약하거나 전혀 사용되지 않을 때도 있다. 남을 위해서뿐아니라 자기자신을 위해서 무엇을

하는가를 설명하는데 습관되도록 하여야 한다. 지금부터 2일동안은 아마 무엇을 했는지 정확히 알지 못할수도 있다.

실행명령문과 Bourne쉘지령 Bourne쉘프로그람은 UNIX지령, Bourne쉘지령, 프로그람작성구조와 설명문의 조합으로 이루어 져 있다.

스크립트를 실행형으로 만들기 파일을 작성할 때 실행허가는 주어 지지 않는다. 스크립트를 실행하기 위해 이 허가가 필요하다. 이때 chmod지령을 사용하여 실행형으로 전환한다.

실례 8-44

1. $ **chmod +x myscript**
2. $ **ls -lF myscript**
   *-rwxr-xr-x 1 ellie 0 Jul 13:00 myscript**

설명

1. chmod지령을 사용하여 사용자, 그룹 등을 위해 실행허가상태로 전환한다.
2. ls지령의 출력은 모든 사용자가 myscript파일에 대해 실행허가를 가지고 있다는것을 표시한다. 파일이름 끝에 있는 별표도 프로그람이 실행형이라는것을 표시한다.

스크립트작성대화조종 다음의 실례에서 사용자는 편집기에서 스크립트를 작성한다. 파일을 보존한 다음 실행허가상태가 설정되고 스크립트가 실행된다. 프로그람에 오유가 있으면 쉘이 즉시에 응답한다.

실례 8-45

(The Script)
% **cat greetings**
1. #!/bin/sh
2. *# This is the first Bourne shell program of the day.*
   # Scriptname: greetings
   # Written by: Barbara Bom
3. echo “Hello $LOGMAME, it's nice talking to you.”
4. echo “Your present working directory is 'pwd'.”
   echo “You are working on a machine called 'uname -n'.” echo “Here is a list of your files.”
5. ls *# List files in the present working directory*
6. echo “Bye for now $LOGNAME. The time is 'date +%T'!”

(The Command Line)
$ **chinod +x greetings**
$ **greetings**
3. *Hello barbara, it's nice talking to you.*
4. *Your present working directory is /home/lion/barbara/prog You are working on a machine called lion.*
   Here is a list of your files.

5. *Afile* *cplus* *letter* *prac*
   Answerbook *cprog* *library* *prac1*
   bourne *joke* *notes* *per15*
6. *Bye for now barbara. The time is 18:05:07!*

설명

1. 스크립트의 첫번째 행 #!/bin/sh는 핵심부에게 이 프로그람에서 어느 해석프로그람이 행들을 실행시키는가(이 경우에는 sh(Bourne쉘)해석프로그람)를 알려 준다.
2. 설명문은 표식 #이 앞에 붙은 실행되지 않는 행이다. 그것들은 행우에 단독으로 놓일수도 있고 지령뒤에 첨부될수도 있다.
3. 쉘에 의해 변수바꿔넣기가 진행된후 echo지령은 현시장치에 행을 현시한다.
4. 쉘에 의해 지령바꿔넣기가 진행된후 echo지령은 현시장치에 행을 현시한다.
5. ls지령이 실행된다. 설명문은 쉘에 의해 무시된다.
6. echo지령은 2중인용부호안에 있는 문자렬을 현시한다. 변수바꿔넣기와 지령바꿔넣기(거꿀인용부호)는 2중인용부호안에서도 진행된다. 이 경우 인용부호가 실제로는 필요 없다.

2. 사용자입력읽기

read지령은 말단이나 파일로부터 입력을 읽어 들이기 위해 사용하는 내부지령이다.

read지령은 행바꾸기가 나타날 때까지 입력행을 읽어 들인다. 행의 끝에 있는 행바꾸기는 읽어 질 때 빈바이트로 번역되게 된다. read지령을 사용하여 사용자가 행바꾸기를 입력할 때까지 프로그람이 중지되게 할수 있다. read지령이 파일로부터 입력행들을 읽어 들이는데 얼마나 효과적으로 사용되는가 하는것은 258페지의 《순환지령》에서 설명하였다.

표 8-11. read지령

형식	의미
Read answer	표준입력으로부터 한개 행을 읽어서 변수 answer에 대입한다.
Read first last	표준입력으로부터 첫번째 공백이나 행바꾸기까지 행을 읽어 들여 첫번째로 입력된 단어는 변수 first에, 나머지는 변수 last에 대입한다.

실례 8-46

(The Script)
$ **cat nosy**
#!/bin/sh
 # Scriptname: nosy
echo "Are you happy? \ c"
1. **read answer**
echo "$answer is the right response."
echo "What is your full name? \ c"

```
2.  read first middle last
     echo "Hello $first"
-----------------------------------------------------------
 (The Output)
    $ nosy
    Are you happy? Yes
1.  Yes is the right response.
2.  What is your full name? Jon Jake Jones
    Hellow Jon
```

설명

1. read지령은 사용자입력행을 받아서 변수 answer에 대입한다.
2. read지령은 사용자로부터 입력을 받아 첫번째 입력단어는 변수 first에, 두번째 단어는 변수 middle에, 나머지단어들은 모두 변수 last에 대입한다.

실례 8-47

```
(The Script)
$ cat printer_check
    #!/bin/sh
    # Scriptname: prin.ter_check
    # Script to clear a hung up printer for SVR4
1.  if [ $LOGNAME != root ]
    then
        echo "Must have root privileges to run this program"
        exit 1
    fi
2.  cat << EOF
    Warning: All jobs in the printer queue will be removed. Please turn off the
    printer now. Press return when you are ready to continue. Otherwise press
    Control C. EOF
3.  read ANYTHING          # Wait until the user turns off the printer
    echo
4.  /etc/init.d/lp stop     # Stop the printer
5.  rm -f /var/spool/lp/SCHEDLOCK /var/spool/lp/cemp*
    echo
    echo "Please turn the printer on now."
6.  echo "Press return to continue"
7.  read ANYTHING            # Stall until the user turns the printer
                             # back on
    echo                     # A blank line is printered
8.  /etc/init.d/lp start     # Start the printer
```

설명

1. 사용자가 뿌리인가를 검사하고 아니면 오유를 보내고 탈퇴한다.

2. here문서를 작성한다. 경고통보가 현시장치에 현시된다.
3. read지령이 사용자입력을 대기한다. 사용자가 행바꾸기건을 누를 때 변수 ANYTHING에 입력되는 모든것이 대입된다. 이 변수가 더이상 사용되지 않는다. 이 경우 read지령은 사용자가 인쇄기를 끄고 다시 행바꾸기건을 누를때까지 기다리는데 사용된다.
4. lp프로그람은 인쇄기데몬을 중지시킨다.
5. 계획작성프로그람이 다시 기동하기전에 /var/spool/lp에 있는 림시파일뿐아니라 SCHEDLOCK파일이 지우기되여야 한다.
6. 사용자에게 준비됐으면 행바꾸기건을 누르겠는가고 물어 본다.
7. 사용자가 입력하는것이 변수 ANYTHING에 들어 가고 행바꾸기건이 눌리우면 프로그람을 다시 실행한다.
8. lp프로그람이 인쇄기데몬들을 기동한다.

3. 산수연산

산수연산은 Bourne쉘에 내장되여 있지 않다. 만일 간단한 옹근수산수연산을 할 필요가 있으면 Bourne쉘스크립트에서는 UNIX의 expr지령을 자주 사용한다. 류동소수점연산을 위해서는 awk나 bc프로그람이 사용될수 있다. 산수연산이 내장되여 있지 않으므로 연산을 여러번 기동하면 쉘의 성능이 떨어 지게 된다. 순환지령에서 계수기가 증가하거나 감소할 때마다 산수연산을 처리하기 위해 또 다른 프로쎄스를 작성할 필요가 있다.

옹근수산수연산과 expr지령 expr지령은 식처리프로그람이다. 이 지령은 연산식을 평가할 때 간단한 옹근수연산들을 진행할수 있다(표 8-12). 이것의 매 인수들은 공백으로 분리해야 한다. +, -, *, /, %연산자들이 제공되는데 묶음 및 우선권과 같은 일반적인 프로그람작성규칙이 적용된다.

표 8-12. expr지령산수연산자

연산자	기 능
* / %	곱하기, 나누기, 나머지
+ -	더하기, 덜기

실례 8-48

1. $ **expr 1 + 4**
   5
2. $ **expr 1+4**
   1+4
3. $ **expr 5 +9 / 3**
   8
4. $ **expr 5 * 4**
   expr: syntax error
5. $ **expr 5 \ * 4 - 2**

18

6. $ **expr 11 % 3**
   2

7. $ **num=1**
   $ **num= `expr $num + 1 `**
   $ **echo $num**
   2

설명

1. expr지령은 식을 평가한다. 2개의 수가 더해 진다.
2. 연산자사이에 공백이 없으므로 식은 문자렬로 평가된다.
3. 더하기와 나누기가 결합된다. 처음에 나누기가 진행되고 그다음에 더하기가 진행된다.
4. 별표(*)는 쉘에 의해 통용기호로 평가되여 expr지령이 실패하게 된다.
5. 별표(*)는 거꿀빗선에 의해 쉘로부터 해석되지 않는다. expr지령이 산수연산을 진행한다.
6. 나머지연산자(%)가 나누기가 끝난 다음 그 나머지를 되돌린다.
7. 변수 num에 1이 대입된다. expr지령은 변수의 값에 1을 더하여 그 결과를 num에 넣는다. num의 값은 현시장치에 출력된다.

류동소수점연산 bc, awk, nawk와 같은 편의프로그람들은 더 복잡한 계산을 할 필요가 있을 때 쓸모가 있다.

실례 8-49

(The Command Line)

1. $ **n='echo "scale=3; 13 / 2" | bc'**
   $ **echo $n**
   6.500
2. **product= `nawk -v x=2.45 -vy=3.123 'BEGIN{printf "%.2f\ n",x*y}' `**
   $ **echo $product**
   7.65

설명

1. echo지령의 출력은 bc프로그람에 파이프된다. scale이 3으로 설정되는데 이 수자는 인쇄될 때 소수점 오른쪽으로 찍히는 수들의 개수이다. 계산은 13을 2로 나누는것이다. 전체 파이프행은 거꿀인용부호로 닫겨 진다. 지령바꿔넣기가 진행되고 출력이 변수 n에 대입된다.
2. nawk프로그람이 지령행에서 넘겨 진 인수목록으로부터 값들을 얻는다. x=2.45, y=3.123이다(-v추가선택은 awk가 아니라 nawk에 대응한다.). 수들이 곱해 진 다음에 printf함수가 그 결과를 소수점아래 두자리까지 형식화하여 인쇄한다. 출력은 변수 product에 대입된다.

4. 위치파라메터와 지령행인수

정보는 지령행을 거쳐 스크립트에 전달될수 있다. 스크립트이름뒤에 있는 매 단어(공백으로 분리된)를 인수라고 한다.

지령행인수들은 스크립트들에서 위치파라메터로 참조될수 있다. 실례로 $1는 첫번째 인수, $2는 두번째 인수, $3은 세번째 인수 등으로 표시할수 있다. $#변수는 파라메터들의 개수를 검사하는데 사용되고 $*는 그것들 모두를 현시하는데 사용된다. 위치파라메터들은 set지령으로 설정 및 재설정될수 있다. set지령이 사용되면 이미 설정된 임의의 위치파라메터들은 지워 진다(표 8-13).

표 8-13. 위치파라메터

위치파라메터	무엇을 참조하는가
$0	스크립트이름을 참조한다.
$#	위치파라메터들의 수를 취한다.
$*	모든 위치파라메터들을 현시한다.
$@	2중인용부호안에 닫겨 있을 때를 제외하고 $*와 같다.
“$*”	한개 인수로 확장된다(실례 “$1 $2 $3”).
“$@”	분리된 인수들로 확장된다(실례 “$1” “$2” “$3”).
$1...$9	9개의 위치파라메터들을 다 참조한다.

실례 8-50

(The Script)

```
    #!/bin/sh
    # Scriptname: greetings
    echo "This script is called $0."
1.  echo "$0  $1 and $2"
      echo "The number of positional parameters is $#"
      -------------------------------------------
(The Command Line)
    $ chmod +x greetings
2.  $ greetings
    This script, is called greetings.
    greetings   and
    The number or positional parameters is
3.  $ greetings Tommy
    This script is called greetings.
    greetings Tommy and
    The number of positional parameters 1
4.  $ greetings Tommy   Kimberly
```

This script is called greetings.
greetings Tommy and Kimberly
The number or positional parameters is 2

설명

1. 스크립트 greetings에서 위치파라메터 $0은 스크립트이름을 참조하고 $1은 첫번째 지령행인수를, $2는 두번째 지령행인수를 참조한다.
2. greetings스크립트는 임의의 인수넘김이 없이 실행된다. 출력은 이 스크립트를 greetings라고 부른다는것과 $1과 $2에는 아무값도 대입되지 않았다는것을 나타낸다. 따라서 그 값들은 빈값이고 아무것도 인쇄되지 않는다.
3. 이때 한개의 인수 Tommy가 넘겨 진다. Tommy는 위치파라메터 1에 대입된다.
4. 2개의 인수 Tommy와 kimberly가 입력된다. Tommy는 $1에, kimberly는 $2에 대입된다.

set지령과 위치파라메터 인수를 가진 set지령은 위치파라메터들을 재설정한다.[4] 한번 재설정되면 이전의 파라메터목록이 없어 진다. 모든 위치파라메터들을 해제하기 위해 set - - 를 사용한다. $0은 언제나 스크립트의 이름이다.

실례 8-51

(The Script)

```
    $ cat args
    #!/bin/sh
    # Scriptname: args
    # Script to test command line arguments
1.  echo The name of this script is $0.
2.  echo The arguments are $*.
3.  echo The first argument is $1.
4.  echo The second argument is $2.
5.  echo The number of arguments is $#.
6.  oldargs=$*              #Save parameters passed in from the
                            # command line
7.  set Jake Nicky Scott    # Reset the positional parameters
8.  echo All the positional parameters are $*.
9.  echo The number of postional parameters is $#.
10. echo "Good-bye for now, $1 "
11. set 'date'              # Reset the positional parameters
12. echo The date is $2 $3, $6.
13. echo "The value of \ $oldargs is $oldargs."
14. set $oldargs
15. echo $1 $2 $3
```

4. 인수가 없는 set 지령은 이 쉘에 대하여 설정된 모든 변수들 즉 국부와 넘겨 진 변수들을 현시한다는것을 명심하여야 한다. 스위치가 있는 set 지령은 -x 와 -v 와 같은 쉘조종스위치들을 절환할수 있다.

(The Output)

$ **args a b c d**

1. *The name* or *this script is **args.***
2. *The arguments are* **a b c d.**
3. *The first argument is **a.***
4. *The second argument is **b.***
5. *The number of arguments is **4.***
8. *All the positional parameters are **Jake Nicky Scott.***
9. *The number of positional parameters is 3.*
10. *Good-bye for now, **Jake***
12. *The date is Mar 25, 2001.*
13. *The value of $oldargs is a b c d.*

설명

1. 스크립트이름이 $0변수에 기억된다.
2. $*는 모든 위치파라메터들을 표시한다.
3. $1은 첫번째 위치파라메터(지령행인수)를 표시한다.
4. $2는 두번째 위치파라메터를 표시한다.
5. $#는 위치파라메터들(지령행인수들)의 총수이다.
6. 모든 위치파라메터들은 oldargs변수에 보존된다.
7. set지령은 낡은 목록을 지우고 위치파라메터들을 재설정한다. 이때 $1은 Jake, $2는 Nicky, $3은 Scott이다.
8. $*는 모든 파라메터들 Jake, Nicky, Scott를 표시한다.
9. $#는 파라메터들의 개수 3을 표시한다.
10. $1는 Jake이다.
11. 지령바꿔넣기가 진행된후 즉 date가 실행된후 위치파라메터는 date지령의 출력으로 재설정된다.
12. $2, $3, $6의 새로운 값들이 현시된다.
13. oldargs에 기억된 값들이 인쇄된다.
14. set지령은 oldargs에 보존된 값으로부터 위치파라메터들을 작성한다.
15. 첫번째 3개의 위치파라메터들이 현시된다.

실례 8-52

(The Script)

```
$ cat checker
#!/bin/sh
# Scriptname: checker
# Script to demonstrate the use of special variable
# modifiers and arguments
1.  name=${1:?"requires an argument" }
    echo Hellow $name
```

(The Command Line)

2. $ **checker**
   ./checker: 1: requires an argument
3. $ **checker Sue**
   Hello Sue

설명

1. 특수변수변경자 :?는 $1이 값을 가지는가를 검사한다. 만일 그렇지 않으면 스크립트는 탈퇴하고 통보문이 인쇄된다.
2. 프로그람이 인수없이 실행된다. $1에는 값이 대입되지 않는다. 오유가 현시된다.
3. checker프로그람에 지령행 인수 Sue가 주어 진다. 스크립트에서 $1에 Sue가 대입된다. 프로그람이 계속된다.

$*와 $@가 어떻게 다른가 $*와 $@는 2중인용부호로 닫길 때만 다르다. $*가 2중인용부호안에 있을 때 파라메터목록은 한개의 문자렬로 된다. $@가 2중인용부호안에 있을 때 매 파라메터들은 각각 인용부호안에 놓인다. 즉 매 단어는 분리된 문자렬로 취급된다.

실례 8-53

1. $ **set 'apple pie' pears peaches**
2. $ **for i in $***
   > do
   > echo $i
   > done
   apple
   pie
   pears
   peaches

3. $ **set 'apple pie' pears peaches**
4. $ **for i in "$*"**
   > do
   > echo $i
   > done
   apple pie pears peaches
5. $ **set 'apple pie' pears peaches**
6. $ **for i in $@**
   > do
   > echo $i
   > done
   apple
   pie
   pears
   peaches

```
7.  $ set 'apple pie' pears peaches
8.  $ for i in "$@"            #At last ! !
    > do
    > echo $i
    > done
    apple pie
    pears
    peaches
```

설명

1. 위치파라메터들이 설정된다.
2. $*이 확장될 때 apple pie를 둘러 싼 인용부호가 벗겨 진다. apple과 pie는 2개의 분리된 단어들로 된다. for순환은 매 단어들을 차례로 변수에 설정한 다음 i의 값을 인쇄한다. 매번 순환될 때마다 왼쪽에 있는 단어가 옮겨 지고 다음단어가 변수 i에 설정된다.
3. 위치파라메터들이 설정된다.
4. $*를 2중인용부호안에 넣어 전체 파라메터목록이 한개 문자렬 "apple pie pears peaches"로 된다. 전체 목록이 한개 단어로 i에 대입된다. 순환은 한번 반복한다.
5. 위치파라메터들이 설정된다.
6. 인용부호가 없는 $@와 $*는 같은 기능을 수행한다(행 2).
7. 위치파라메터들이 설정된다.
8. $@를 2중인용부호안에 넣어 매 위치파라메터가 인용부호안에 있는 문자렬로 처리된다. 목록은 "apple pie", "pears", "peaches"로 된다. 요구되는 결과가 최종적으로 얻어 진다.

5. 조건지령들과 흐름조종

조건지령들은 조건이 맞는가 맞지 않는가에 따라 과제들을 수행할수 있게 한다. if지령은 결심채택의 가장 간단한 형식이다. if/else지령은 두가지 판단을 할수 있게 하고 if/elif/else지령은 다중판단을 할수 있게 한다.

Bourne쉘에서는 if다음에 지령이 놓여야 한다. 지령은 체계지령이나 내부지령일수 있다. 지령의 탈퇴상태를 사용하여 조건을 평가한다.

식을 평가하기 위해서 내부지령 test지령을 사용한다. 이 지령은 또한 중괄호로 표시할수 있다. test지령에 사용되는 식이 단일중괄호안에 놓여 사용될수 있다. 쉘메타문자(통용기호)들은 test지령에 의해 확장되지 않는다. 지령의 결과가 검사되여 탈퇴상태가 령이면 성공을 현시하고 령이 아니면 실패를 현시한다(표 8-14).

탈퇴상태시험 다음의 실례들은 탈퇴상태가 검사되는 방법을 보여 준다.

실례 8-54

(At the Command Line)

```
1.  $ name=Tom

2.  $ grep "$name"   /etc/passwd
    Tom: 8ZKX2F:5102:40:Tom Savage:/HOME/TOM: /bin/ksh
3.  $ echo $?
    0               Success !
4.  $ test $name != Tom
5.  $ echo $?

    1               Failure
6.  $ [ $name = Tom ]       #Brackets replace the test command
7.  $ echo $?
    0               Success
8.  $ [ $name = [Tt]?m ]          #Wildcards are not evaluated
9.  $ echo $?                           # by the test command
    1
```

설명

1. 변수 name에 문자렬 Tom이 대입된다.
2. grep지령은 passwd파일에서 문자렬 Tom을 탐색한다.
3. ?변수에는 실행되는 마지막지령의 탈퇴상태가 들어 있는데 이 경우 grep의 탈퇴상태가 들어 있다. grep가 문자렬 Tom을 탐색하는데서 성공하면 탈퇴상태 0을 되돌린다. grep지령은 성공했다.
4. test지령은 문자렬과 수들을 평가하고 파일시험을 하는데 사용된다. 모든 지령들과 같이 그것은 탈퇴상태를 되돌린다. 탈퇴상태가 령이면 식이 참이고 탈퇴상태가 1이면 식이 거짓이라고 평가된다. 같기부호주위에 공백이 있어야 한다. name의 값이 Tom과 같지 않는가를 검사한다.
5. 검사가 실패하여 탈퇴상태를 1로 되돌린다.
6. 중괄호는 test지령을 대신하는 표기법이다. 첫번째 중괄호다음에 공백이 있어야 한다. 식이 검사되여 $name이 Tom과 같은가를 본다.
7. 검사의 탈퇴상태가 령이다. $name이 Tom과 같기때문에 검사가 성공했다.
8. test지령에서는 통용기호를 사용할수 없다. 물음표가 문자그대로 취급되므로 검사가 실패한다. Tom과 [Tt]?m는 같지 않다.
9. 탈퇴상태는 1로 되여 8행의 본문이 거짓이라는것을 보여 준다.

test지령 test지령은 조건식을 평가하여 참 또는 거짓을 되돌리는데 사용한다. 참일 때 탈퇴상태를 0, 거짓일 때 령이 아닌 값을 되돌린다. test지령이나 중괄호가 사용될수 있다.

표 8-14. 문자렬, 옹근수와 파일시험

검사연산자	무엇을 검사하는가
문자렬검사:	
string1 = string2	string1과 string2가 같다(같기주위에 공백이 있어야 한다.).

(표계속)

검사연산자	무엇을 검사하는가
string1 != string2	string1과 string2 가 같지 않다(!=주위에 공백이 있어야 한다.).
string	string이 빈값이 아니다.
-z string	string의 길이가 령이다.
-n string	string의 길이가 령이 아니다.
실례 :	test –n $word 또는 [-n $word] test tom = sue 또는 [tom = sue]
옹근수검사:	
int1 -eq int2	int1은 int2와 같다.
int1 -ne int2	int1은 int2와 같지 않다.
int1 -gt int2	int1은 int2보다 크다.
int1 -ge int2	int1은 int2보다 크거나 같다.
int1 -lt int2	int1은 int2보다 크다.
int1 -le int2	int1은 int2보다 작거나 같다.
파일시험:	
-b filename	블로크특수파일
-c filename	문자특수파일
-d filename	등록부존재
-f filename	정규파일은 존재하고 등록부는 존재하지 않는다.
-g filename	묶음설정 ID가 설정된다.
-k filename	sticky비트가 설정된다.
-p filename	파일은 이름 지어 진 파이프이다.
-r filename	파일을 읽을수 있다.
-s filename	파일의 크기는 0이 아니다.
-u filename	사용자설정 ID비트가 설정된다.
-w filename	파일을 쓸수 있다.
-x filename	파일을 실행할수 있다.

if지령 조건지령의 가장 간단한 형식이 if지령이다. if다음에 오는 지령이나 UNIX 편의프로그람이 실행되여 그 탈퇴상태가 복귀된다. 탈퇴상태는 편의프로그람을 작성하는 프로그람작성자에 의해서 결정된다. 만일 탈퇴상태가 0이면 지령은 성공하고 then단어다음에 오는 명령문이 실행된다. C쉘에서 if지령다음에 오는 식은 C에서와 같이 론리형식의 식이다. 그러나 Bourne쉘과 Korn쉘에서는 if다음에 오는 명령이 지령이거나 지령들의 묶음이다. 평가될 지령의 탈퇴상태가 0이면 then다음에 오는 명령묶음은 fi가 발견될 때까지 실행된다. fi는 if명령블로크를 끝낸다.

탈퇴상태가 0이 아닌 경우 지령이 실패했다는것을 의미하는데 then다음에 있는 명령이 무시되고 조종이 fi명령다음의 행으로 직접 넘어 간다. 검사될 지령의 탈퇴상태를 아는것이 중요하다. 실례로 grep의 탈퇴상태는 grep가 파일에서 찾고 있는 패턴을 찾았는가를 확고하게 알수 있게 한다. grep가 탐색에서 성공하면 0탈퇴상태를 넘긴다. 성공하지 못하면 령이 아닌 값을 넘긴다. sed와 awk프로그람도 패턴을 탐색하지만 그것을 찾았는가는 고려하지 않고 성공한 탈퇴상태만 알려 준다. sed와 awk에서 성공을 가르는 기준은 기능이 아니라 정확한 문법이다.[5]

형식

```
if 지령
then
        지령
        지령
fi
------------------------------------
if test expression
then
        지령
       fi
                          or
if [ expression]
then
       지령
fi
-----------------------------------
```

실례 8-55

```
1.  if ypmatch "$name" passwd > /dev/null 2>&1
2.  then
            echo Found $name!
3.  fi
```

설명

1. ypmatch지령은 써버콤퓨터의 NIS Passwd자료기지에서 자기의 인수 name을 탐색하는 NIS(Suns Network Information Services)지령이다. 표준출력과 표준오유가 UNIX비트바키트 /dev/null에로 방향바꾸기된다. 만일 ypmatch가 사용자의 체계에서 지원되지 않으면 다음과 같이 해야 한다.
   if grep "name"/etc/passwd > /dev/null 2>&1
2. ypmatch지령의 탈퇴상태가 령이면 프로그람은 then명령으로 이행하여 fi가 발견될 때까지 지령들을 실행한다.
3. fi는 then명령다음에 오는 지령들의 목록의 끝을 알려 준다.

---

5. C쉘과 달리 Bourne쉘은 간단한 명령에 대해서도 then이 없는 if명령문을 지원하지 않는다.

실례 8-56

```
1.  echo  "Are you okay (y/n) ?"
    read answer
2.  if [ "$answer" = Y –o "$answer" = y ]
    then
        echo  "Glad to hear it"
3.  fi
```

설명

1. 사용자에게도 질문이 제기되여 응답이 요구된다. read지령은 응답을 대기한다.
2. 중괄호로 표시되는 test지령은 식을 검사하는데 사용된다. 식이 옳으면 탈퇴상태를 0으로 되돌리고 옳지 않으면 0이 아닌 탈퇴상태를 되돌린다. 만일 변수 answer가 Y나 y와 같으면 then다음 명령이 실행된다(test지령은 식을 검사할 때 통용기호의 사용을 허용하지 않으며 중괄호주변에 공백이 있어야 한다. =연산자주위에도 공백이 있어야 한다.). 표 8-14를 참고하기 바란다. $answer를 한개 문자렬로 하기 위해 2중인용부호안에 넣는다. test지령은 한개이상의 단어가 =연산자앞에 나타나면 실패한다. 실례로 사용자가 yes, you betcha를 입력할 때 $answer가 2중인용부호안에 넣어 지지 않으면 answer변수의 값이 3개 단어로 되여 test지령이 실패한다.
3. fi는 then명령다음에 오는 지령들의 목록의 끝을 표시한다.

exit지령과 ?변수 exit지령을 사용하여 스크립트를 끝내고 지령행으로 복귀한다. 만일 어떤 조건이 생기면 스크립트가 탈퇴할것을 요구할수 있다. exit지령에 주어 지는 인수의 값 범위는 0으로부터 255까지이다. 프로그람이 인수를 령으로 하여 탈퇴할 때 프로그람은 성공하여 탈퇴한다. 인수가 령이 아닐 때는 어떠한 실패를 의미한다. exit지령에 주어 지는 인수는 쉘의 ?변수에 기억된다.

실례 8-57

```
(The Script)
$ cat bigfiles
    # Name: bigfiles
    #Purpose Use the find command to find any files in the root
    #partition that have not been modified within the past n (any
    #number within 30 days) days and are larger than 20 blocks
    # (512-byte blocks)

1.  if  [ $# -ne 2 ]
    then
        echo  "Usage:  $0 mdays size  " 1>&2
        exit 1
2.  fi
3.  if  [ $1 -lt 0 -o $1 -gt 30 ]
    then

        echo "mdays is out of range"
```

```
        exit 2
4.  fi
5.  if [ $2 -le 20 ]
    then
        echo    size is out of range
        exit 3
6.  fi
7.  find / -xdev -mtime $1 -size +$2 -print
```

(The Command Line)
$ **bigfiles**
Usage: bigfiles mdays size

$ **echo $?**
1

$ **bigfiles 400 80**
mdays is out of range

$ **echo $?**
2

$ **bigfiles 25 2**
size is out of range   »

$ **echo $?**
3

$ **bigfiles 2 25**
(Output of find prints here)

설명

1. 이 명령은 다음과 같다. 만일 인수들의 수가 2와 같지 않으면 오유통보를 인쇄하고 그것을 표준오유에 보낸 다음 탈퇴상태를 1로 하여 스크립트를 탈퇴한다.
2. fi는 then다음의 명령블로크의 끝을 표시한다.
3. 명령은 다음과 같다. 지령행으로부터 넘어 온 위치파라메터의 값이 0보다 작거나 30보다 더 크면 통보를 인쇄하고 상태를 2로 하여 탈퇴한다. 표 8-14에서 수연산자들을 참고하기 바란다.
4. fi는 if블로크를 끝낸다.
5. 지령행에서 넘어 온 위치파라메터의 값이 20(512byte묶음들)보다 작거나 같으면 통보를 인쇄하고 상태를 3으로 하여 탈퇴한다.
6. fi는 if블로크를 끝낸다.
7. find지령이 홈등록부에서 탐색을 시작한다. -x dev추가선택은 find지령이 다른 부분들을 탐색하지 않도록 한다. -mtime추가선택은 파일이 수정된 때로부터 지나간 날자수를 현시하는 수인수를 취하고 -size추가선택은 512byte블로크에서 파일의 크기를 나타내는 수인수를 취한다.

NULL**값검사** 변수의 빈값을 검사할 때 2중인용부호를 사용하여 빈값을 표시해야 한다. 그렇지 않으면 test지령이 실패한다.

실례 8-58

(The Script)

```
1.  if [ "$name" = "" ]
        #Alternative to [ ! "$name" ] or [-z "$name" ]
           then
        echo The name variable is null
    fi
```

(From System showmount program, which displays all remotely mounted systems)

```
    remotes='/usr/sbin/showmount'
2.  if [ "-X${reimotes}" !="X" ]
    then
         /usr/sbin/wall ${remotes}
               ...
3.  fi
```

설명

1. name변수가 빈값이면 그 시험은 성공한다. 2중인용부호를 사용하여 빈값을 표시한다.
2. showmount지령은 주콤퓨터로부터 떨어 진 모든 의뢰자들을 표시한다. 이 지령은 한개 또는 그이상의 의뢰자들을 표시하거나 아무것도 표시하지 않을수 있다. 변수 remotes는 대입된 값을 가지거나 빈값을 가질수 있다. 검사될 때 문자 x가 변수 remotes앞에 놓인다. remotes가 빈값이면 떨어져 있는 그 어느 의뢰자도 등록가입되지 않고 X는 X와 같게 되여 프로그람이 행 3에서 다시 실행을 시작하게 한다. 만일 변수가 값을 가지면 실례로 기본이름 pluto를 가지면 식은 if Xpluto!=X로 되고 wall지령이 실행된다(주콤퓨터와 떨어 진 모든 사용자에게 통보를 보낸다.). 식에서 X를 사용하는 목적은 remotes의 값이 빈값인 경우에 식에서 !=연산자의 어느쪽에든지 언제나 공간을 메꿀수 있게 하자는것이다.
3. fi는 if를 끝낸다.

if/else지령 if/else지령은 2가지 판단을 내릴수 있게 한다. if다음의 지령이 실패하면 else다음지령이 실행된다.

형식

```
if 지령
then
        지령(들)
```

```
else
        지령(들)
fi
```

실례 8-59

(The Script)

```
     #1/bin/sh
1.  if ypmatch "$name" passwd > /dev/null 2>&1 6
2.  then
         echo Found $name!
3.  else
4.       echo   "Can't find $name "
         exit 1
5.  fi
```

설명

1. ypmatch지령은 NIS passwd자료기지에서 인수 name을 탐색한다. 사용자가 출력을 볼 필요가 없으므로 표준출력과 표준오유가 UNIX비트바키트 /dev/null에로 방향바꾸기된다.
2. ypmatch지령의 탈퇴상태가 0이면 프로그람조종이 then명령으로 넘어 가고 else전까지의 지령들을 실행한다.
3. ypmatch지령이 passwd자료기지에서 $name을 찾지 못하면 else명령다음의 지령들이 실행된다. 즉 else블로크안에 있는 지령들이 실행되자면 ypmatch의 탈퇴상태가 0이 되지 말아야 한다.
4. passwd자료기지에 $name의 값이 없으면 echo명령이 실행되며 프로그람은 값을 1로 하여 실패를 알리고 탈퇴한다.
5. fi는 if를 끝낸다.

실례 8-60

(The Script)

```
$ cat idcheck
#!/bin/sh
# Scriptname: idcheck
# purpose: check user ID to see if user is root.
# Only root has a uid 0.
# Format for id oucput:uid=9496(ellie) gid=40 groups=40
# root's uid=0

1.  id=`id | nawk -F'[=()' '{print $2}'`        # Get user ID
    echo your user id is: $id
2.  if [ $id -eq 0 ]
```

---

6. NIS+를 사용하면 지령은 다음과 같다.
   if nismatch "$name"passwd. org_dir

```
        then
3.          echo "you are superuser."
4.  else
            echo "you are not superuser."
5.  fi

    (The Command Line)
6.  $ idcheck
    your user id is; 9496
    you are not superuser.
7.  $ su
    Password ;
8.  # idcheck
    your user id is: 0
    you are superusser
```

설명

1. id지령은 nawk지령에 파이프된다. nawk는 갈기부호와 열린괄호를 마당분리기로 사용하고 출력으로부터 사용자 ID를 얻어 내며 그 출력을 변수 id에 넣는다.
2. id의 값이 0과 같으면 행 3이 실행된다.
4. ID가 0이 아니면 else명령이 실행된다.
5. fi는 if지령의 끌을 나타낸다.
6. idcheck스크립트는 사용자식별번호가 9496인 현재사용자에 의해 실행된다.
7. su지령이 사용자를 뿌리에로 절환시킨다.
8. #재촉문은 주사용자(뿌리)가 새로운 사용자라는것을 알려 준다. 뿌리의 UID는 0이다.

if/elif/else지령 if/elif/else지령은 다중판단을 내릴수 있게 하는 지령이다. if다음 지령이 실패하면 elif의 다음지령이 검사된다. 만일 이 지령이 성공하면 then의 다음지령들이 실행된다. 그러나 이 지령이 실패하면 다음 elif지령이 검사된다. 어느한 지령도 성공하지 못하면 else지령들이 실행된다. else블로크를 기정값이라고 한다.

형식

```
if 지령
then
      지령(들)
elif 지령
then
      지령(들)
elif 지령
then
      지령(들)
else
```

```
        지령(들) :
    fi
```

실례 8-61

```
    (The Script)
    $ cat tellme
        #!/bin/sh
        # Scriptname: telime
    1.  echo -n "How old are you? "
        read age
    2.  if [ $age -lt 0 -o $age -gt 120]
        then
            echo "Welcome to our planet! "
            exit 1
        fi
    3.  if [ $age –ge 0 –a $age –lt 13 ]
        then
            echo "A child is a garden of verses"
        elif [ $age –ge 12 –a $age –lt   20 ]
        then
            echo "Rebel without a cause"
        elif [ $age –gt 20 –a $age –lt   30 ]
        then
            echo "You got the something..."
    4.  else
            echo "Sorry I asked"
    5.  fi

    (The Output)
    $ tellname
    How old are you? 200
    Welcome to our planet!

     $ tellme
    How are you? 13
    Rebel without a cause

     $ tellme
    How old are you? 55
    Sorry I asked
```

설명

1. 사용자입력을 요구한다. 입력이 변수 age에 대입된다.
2. 중괄호안에서 수검사가 진행된다. age가 0보다 작거나 120보다 크면 echo 지령이 실행되고 프로그람은 탈퇴상태를 1로 하고 끝난다. 대화형쉘재촉문이 나타난다.
3. 중괄호안에서 수검사가 진행된다. age가 0보다 크거나 13보다 작으면 test

지령은 탈퇴상태를 0으로 즉 참으로 넘기고 then다음의 명령이 실행된다. 그렇지 않으면 프로그람조종이 elif로 넘어 간다. 그 검사도 실패하면 다음 elif지령이 검사된다.
4. else지령은 기정값이다. 그우의 명령들이 다 실패하면 else지령이 실행된다.
5. fi는 초기 if명령을 끝낸다.

파일시험 보통 스크립트를 작성할 때 스크립트는 사용할수 있는 파일을 요구하며 파일들은 특수허가와 어떠한 형태 또는 다른 속성들을 가져야 한다(표 8-14). 파일시험은 종속적인 스크립트들을 작성하기 위해 필요한 부분이다.

if명령들이 겹싸여 질 때 fi명령은 가장 가까운 if명령을 끝낸다. 겹싸여 진 if명령들을 행들의 시작에서 같은 길이만큼 들여다 쓰면 어느 if명령들이 어느 fi명령들과 겹싸여 지는가를 더 쉽게 알수 있다.

실례 8-62

(The Script)

```
   #!bin/sh
   file=./testing
1. ┌ if [ -d $file ]
   │ then
   │       echo "$file is a directory"
2. │ elif [ -f $file]
   │ then
3. │ ┌ if [ -r $file –a –w $file –a –x $file ]
   │ │ then   # nested if command
   │ │       echo "You have read, write, and execute \
   │ │       permission on $file."
4. │ └ fi
5. │  else
   │     echo "$file is neither a file nor a directory."
6. └ fi
```

설명

1. 파일 testing이 등록부이면 testing is a directory를 인쇄한다.
2. 파일 testing이 등록부가 아니고 보통 파일이면 …
3. 파일 testing이 읽기, 쓰기, 실행가능하면 …
4. fi는 제일 안쪽 if지령을 끝낸다.
5. 행 1, 2가 거짓이면 else지령이 실행된다.
6. 이 fi가 첫번째 if를 끝낸다.

null지령 옹근두점으로 표시되는 null지령은 령을 탈퇴상태로 되돌리고 아무것도 하지 않는 내부지령이다. 이 지령은 사용자가 아무것도 할것이 없지만 어떤 지령이 필요하거나 또는 then명령다음에 프로그람이 무엇인가를 요구하기때문에 오유통보를 내보낼 때 if지령다음에 자리를 채우기 위해 사용된다. 보통 null지령은 순환을 무한순환으로 되게 하기 위하여 순환지령의 한개 인수로 사용된다.

실례 8-63

(The Script)
```
1.  naine=Tom
2.  if grep "$name- databasefile > /dev/null 2>&1
    then
3.       :
4.  else
        echo "$1 not found in databasefiles"
        exit 1
    fi
```

설명

1. 변수 name에 문자렬 Tom이 대입된다.
2. if지령은 grep지령의 탈퇴상태를 검사한다. Tom이 자료기지파일에서 발견되면 옹근두점인 null지령이 실행되여 아무것도 하지 않는다.
3. 옹근두점은 null지령이다. 그것은 다만 탈퇴상태를 0으로 되돌린다.
4. 실제로 하려고 하는것은 오유통보를 인쇄하고 Tom이 발견되지 않으면 탈퇴하는것이다. grep지령이 실패하면 else다음의 지령들이 실행된다.

실례 8-64

(The Command Line)
```
1.  $ DATAFILE=
2.  $ : ${DATAFILE:=$HOME/db/datafile}
    $ echo $DATAFILE
    /home/jody/ellie/db/datafile:
3.  $ : ${DATAFILE:=$HOME/junk}
    $ echo $DATAFILE
    /home/jody/ellie/db/datafile
```

설명

1. 변수 DATAFILE에 빈값이 대입된다.
2. 옹근두점지령은 아무것도 하지 않는 지령이다. 변경자(:=)는 변수에 대입될수 있는 값이나 검사에 사용될수 있는 값을 되돌린다. 이 실례에서 형식이 인수로서 아무것도 하지 않는 지령에 넘겨 진다. 쉘은 변수바꿔넣기를 진행한다. 즉 DATAFILE이 이미 값을 가지지 않으면 DATAFILE에 경로이름을 넣는다. 변수 DATAFILE이 항구적으로 설정된다.
3. 변수가 이미 설정되여 있으므로 변경자오른쪽에 있는 기정값으로 재설정되지 않는다.

실례 8-65

(The Script)

```
$ cat wholenum
     #!bin/sh
1.   # Name:wholenum
     # Purpose:The expr coamand tests that the user enters an integer
     echo "Enter a number."
     read number
2.   if expr "$number" + 0 > /dev/null 2>&1
     then
3.        :
     else
4.        echo "You did not enter an integer value."
          exit 1
5.   fi
```

설명

1. 사용자에게 옹근수를 입력할것을 요구한다. 수가 변수 number에 대입된다.
2. expr지령이 식을 평가한다. 더하기가 진행될수 있다면 수는 전체 수로 되고 expr는 성공을 표시하는 탈퇴상태를 되돌린다. 모든 출력이 비트바키트 /dev/null에로 방향바꾸기된다.
3. expr가 성공하면 탈퇴상태를 령으로 되돌리고 옹근두점지령은 아무것도 하지 않는다.
4. expr지령은 실패하면 령이 아닌 탈퇴상태를 되돌리고 echo지령은 통보를 현시하며 프로그람은 탈퇴한다.
5. fi는 if블로크를 끝낸다.

6. case지령

case지령은 if/elif지령대신에 사용되는 다중분기지령이다. case변수의 값은 값1, 값2 등과 대응되는것이 있을 때까지 비교된다. 어느 한 값이 case변수와 같으면 값다음의 지령은 2중반두점이 나타날 때까지 실행된다. 그다음에 단어 esac(case를 거꾸로 쓴것)다음의 지령이 실행된다.

case변수와 같은것이 없으면 프로그람은 ;;나 esac가 나타날 때까지 기정값 *)다음의 지령들을 실행한다. 기정값 *)값의 기능은 if/else조건지령에서 else명령과 같다. case값으로는 쉘통용기호와 두 값중 하나를 선택할수 있는 수직선기호(파이프기호)를 사용할수 있다.

형식

```
case variable in
value1)
     지령(들)
     ;;
value2)
     지령(들)
```

```
        ;;
    *)
    지령(들)
          ;;
    esac
```

실례 8-66

```
(The Script)
$ cat colors
    #! / bin/ sh
    # Scriptname: colors
1.  echo -N "Which color do you like?"
                read color
2.  case "$color" in
3.  [Bb]l??)
4.      echo I feel $color
        echo The sky is $color
5.      ;;
6.  [Gg]ree*)
        echo   $color is for trees
        echo   $color is for seasick;;
7.  red | orange)               # The vertical bar means "or"
        echo   $color is very warm!;;
8.  *)
        echo   No such color as $color;;
9.  esac
10. echo   "Out of case"
```

설명

1. 사용자입력이 요구된다. 입력이 변수 color에 대입된다.
2. case지령이 표현 $ color를 평가한다.
3. color의 값이 B나 b로 시작하여 그다음에 문자 l과 임의의 2개 문자가 오면 case식은 첫번째 값과 대응된다. 이 값은 한개의 닫긴 괄호로 끝난다. 통용기호는 파일이름확장에 사용되는 메타문자이다.
4. 행번호 3에 있는 값이 case식과 맞으면 그 명령이 실행된다.
5. 이 지령블로크의 마지막지령뒤에 2중반두점이 있어야 한다. 반두점이 나타날 때 조종이 행 10으로 넘어 간다. 반두점들이 그것들의 행우에 있으면 스크립트를 오유수정하기가 더 쉽다.
6. case식이 G나 g로 시작되여 문자 ree가 놓이고 그다음에 임의의 수의 문자들로 끝나는 값과 맞으면 echo지령이 실행된다. 2중반두점이 명령블로크를 끝내고 조종은 10행으로 넘어 간다.
7. 수직선은 or조건연산자로 사용된다. case식이 red나 orange와 맞으면 echo지령이 실행된다.
8. 이것은 기정값이다. 만일 우의 값들중 어느것도 case식과 맞지 않으면 *)

다음의 지령이 실행된다.
9. esac명령이 case지령을 끌낸다.
10. case값들중 하나가 일치되면 여기서 계속 실행된다.

here문서와 case지령을 사용하여 차림표작성 here문서와 case지령이 자주 함께 쓰인다. here문서를 사용하여 현시장치에 현시되는 선택차림표를 만들수 있다. 사용자는 차림표항목들중 하나를 선택할수 있으며 case지령은 선택모임을 검사하여 적합한 지령을 실행한다.

실례 8-67

```
(The .profile File)
    echo "Select a terminal type: "
1.  cat << EMDIT
        1)  vt 120
        2)  wyse50
        3)  sun
2.  ENDIT
3.  read choice
4.  case "$choice" in
5.  1)  TERM=vtl20
        export TERM
        ;;
    2)  TERM=wyse50
        export TERM
        ;;
6.  3)  TERM=sun
        export TERM
        ;;
7.  esac
8.  echo "TERM is $TERM.'

(The Output)
$ . .profile
Select a terminal type:
1) vtl20
2) wyse50
3) sun
3                                   <--User input
TERM is sun.
```

설명

1. 등록가입할 때 .profile에 스크립트의 이 부분이 놓여 있으면 적당한 말단형태를 선택할수 있다. here문서를 사용하여 선택차림표를 현시한다.
2. 사용자정의최종완료자 ENDIT는 here문서의 끝을 현시한다.
3. read지령은 사용자입력을 변수 TERM에 기억시킨다.

4. case지령은 변수 TERM을 평가하여 그 값을 닫긴괄호앞의 값들 즉 1, 2 또는 *들중 어느 한 값과 비교한다.
5. 검사된 첫번째 값이 1이다. 검사가 성공하면 말단으로 vt120이 설정된다. TERM변수가 넘겨 져 보조쉘들이 그것을 계승한다.
6. 기정값이 요구되지 않는다. TERM변수는 보통 등록가입할 때 /etc/profile에서 대입된다. 만일 선택이 3이라면 말단은 sun으로 설정된다.
7. esac는 case지령을 끝낸다.
8. case지령이 끝난후 행이 실행된다.

7. 순환지령

순환지령들은 한개 지령이나 지령묶음을 지정된 회수만큼 또는 어떤 조건이 만족될 때까지 실행시키는데 사용된다. Bourne쉘에는 3가지 형태의 순환 즉 for순환, while순환 그리고 until순환이 있다.

for지령 for순환지령은 항목들의 목록에 대하여 지령들을 유한회수만큼 실행시키는데 사용된다. 실례로 이 순환을 사용하여 파일이나 사용자이름들의 목록에 대하여 같은 지령을 실행할수 있다. for지령다음에 사용자정의된 변수, 예약어 in, 단어들의 목록이 놓인다. 첫번째 순환에서 단어목록의 첫번째 단어가 변수에 대입되고 그다음 목록을 한자리 자리옮김한다. 일단 단어가 변수에 대입되면 순환본체에 들어 가 예약어 do와 done사이에 있는 지령들이 실행된다. 다음번에 순환할 때 두번째 단어가 대입되고 이런 식으로 계속된다. 순환본체는 예약어 do에서 시작하여 done에서 끝난다. 목록에 있는 모든 단어들이 자리옮김되면 순환은 끝나고 프로그람조종이 예약어 done다음의 지령으로 넘어 간다.

형식

```
for variable in word_list
do
     지령(들)
done
```

실례 8-68

(The Script)
$ **cat forloop**
    #!/bin/sh
    # Scriptname: forloop
1. **for pal in Tom Dick Harry Joe**
2. **do**
3.   echo “Hi $pal”
4. **done**
5. echo “Out of loop”

(The Output)
$ **forloop**
Hi *Tom Hi*

Dick Hi
Harry

설명

1. for순환은 이름들의 목록 Tom, Dick, Harry, Joe를 반복하여 사용된 다음 자리옮김한다(왼쪽으로 밀어서 그 값을 사용자정의된 변수 pal에 대입한다.). 모든 단어들이 자리옮김되여 단어목록이 비게 되면 곧 순환이 끝나고 예약어 keyword다음 지령이 실행된다. 첫번째 순환에서 변수 pal에 단어 Tom이 대입된다. 두번째 순환에서는 pal에 Dick가, 다음번에는 Harry가, 마지막에는 Joe가 pal에 대입된다.
2. 예약어 do는 단어목록다음에 놓여야 한다. 만일 이것을 같은 행에 놓는다면 목록을 반두점으로 끝내야 한다.
   실례 for pal in Tom Dick Harry Joe; do
3. 이것은 순환본체이다. Tom이 변수 pal에 대입된후 순환본체에 있는 지령들(즉 do와 done사이에 있는 모든 지령들)이 실행된다.
4. 예약어 done은 순환을 끝낸다. 목록의 마지막단어(Joe)가 대입되여 자리옮김되면 순환이 끝난다.
5. 순환이 끝날 때 조종이 다시 시작된다.

실례 8-69

(The Command Line)
1. $ **cat mylist**
   torn
   patty
   aim
   jake

(The Script)
$ **cat mailer**
   #!/bin/sh
   # Scriptnaae: mailer
2. **for person in `cat mylist`**
   do
3. mail $person < letter
   echo $person was sent a letter.
4. done
5. echo “The letter has been sent.”

설명

1. mylist파일의 내용이 현시된다.
2. 지령바꿔넣기가 진행되여 mylist의 내용이 단어목록으로 된다. 첫번째 순환때 tom이 변수 person에 대입되고 그다음에 자리옮김되여 patty로 교체되고 이런 식으로 계속된다.
3. 순환본체에서 매 사용자에게 letter파일을 보낸다.

4. 예약어 done은 순환반복의 끝을 현시한다.
5. 목록에 있는 모든 사용자에게 우편이 전달되여 순환이 끝나면 이 행이 실행된다.

실례 8-70

(The Script)

```
#!/bin/sh
# Scriptname: backup
# Purpose:
# Create backup files and store them in a backup directory
1.  dir=/home/jody/ellie/backupscripts
2.  for file in memo[1-5]
    do
        if [ -f $file ]
        then
              cp $file $dir/Sfile.bak
              echo "$file is backed up in $dir"
        fi
    done
```

(The Output)

memol is backed up in /home/jody/ellie/backupscripts
memo2 is backed up in /home/jody/ellie/backupscripts
memo3 is backed up in /home/jody/ellie/backupscripts
memo4 is backed up in /home/jody/ellie/backupscripts
memo5 is backed up in /home/jody/ellie/backupscripts

설명

1. 변수 dir가 대입된다.
2. 단어목록은 현재작업등록부에서 memo로 시작되여 1과 5사이의 수자로 끝나는 모든 파일들로 이루어 진다.
3. 순환이 반복될 때마다 한번에 하나씩 매 파일이름이 변수 file에 대입된다. 순환이 진행될 때 파일을 검사하여 그것이 존재하면 실지파일이라는것을 확인한다. 만일 그렇다면 그것을 등록부 /home/jody/ellie/backupscripts에 확장자 bak를 붙여서 복사한다.

단어목록에서 $*와 $@변수 $*와 $@는 확장될 때 2중인용부호속에 닫기지 않으면 같은 의미로 된다. $*는 한개 문자렬로 되고 $@는 개개 단어들의 목록으로 된다.

실례 8-71

(The Script)

```
$ cat greet
  #!/bin/sh
```

```
     # Scriptname: greet
  1. for name in $*          # Same as for name in $@
  2. do
         echo Hi $name
  3. done
  (The Conanand Line)
  $ greet Dee Bert Lizzy Tommy
  Hi Dee
  Hi Bert
  Hi Lizzy
         Hi Tommy
```

설명

1. $*와 $@는 모든 위치파라메터들의 목록 즉 이 경우에는 지령행으로부터 넘겨 진 인수들인 Dee, Bert, Lizzy 그리고 Tommy로 확장된다. 목록에 있는 매 이름들이 for순환에서 변수 name에 차례로 대입된다.
2. 순환본체에 있는 지령들은 목록이 빌 때까지 실행된다.
3. 예약어 done은 순환본체의 끝을 나타낸다.

실례 8-72

```
(The script)
$ cat greet
    #!/bin/sh
    #Scriptname: permx
1.  for file
     do
2.       if  [ -f $file –a ! $file ]
         then
3.               chmod +x $file
                 echo $file now has execute permission
         fi
     done
(The Comman9d Line)
4.  $ permx *
    adden now has execute permission
    checken now has execute permission
    doit now has execute
```

설명

1. for순환에 단어목록이 주어 지지 않으면 그것은 위치파라메터들을 반복한다. 이것은 $*에서 for file과 같다.

2. 지령행으로부터 파일이름들이 들어 온다. 쉘은 별표(*)를 현재작업등록부에 있는 모든 파일이름들로 확장한다. 만일 파일이 보통파일이고 실행허가를 가지고 있지 못하면 3행이 실행된다.
3. 실행허가가 처리되는 매 파일에 추가된다.
4. 지령행에서 별표는 쉘에서 통용기호로 인식되여 현재등록부의 모든 파일들이 *대신 교체된다. 파일들은 인수로서 permx스크립트에 넘겨 진다.

while지령 while지령은 그다음의 지령을 즉시에 평가하여 그것의 탈퇴상태가 령이면 순환본체에 있는 지령들(do와 done사이에 있는 지령들)을 실행시킨다. 예약어 done이 나타나면 조종이 순환의 꼭대기로 다시 올라 가고 while지령이 지령의 탈퇴상태를 다시 검사한다. while에 의해서 평가되는 지령의 탈퇴상태가 령이 아니라고 평가될 때까지 순환이 계속된다. 탈퇴상태가 령이 아니면 예약어 done다음에서 프로그람이 실행된다.

형식

```
while 지령
do
    지령(들)
done
```

실례 8-73

(The Script)
```
$ cat num
    #!/bin/sh
    # Scriptname:num
1.  num=0              # Initialize num
2.  while  [ $num –lt 10 ]              # Test num with test command
    do
       echo  -n $num
3.     num=`expr  $num + 1`              # Increment num
    done
    echo "\ nAfter loop exits, continue running here"
```

(The Output)
```
0123456789
         After loop exits, continue running here
```

설명

1. 이것은 초기화단계이다. 변수 num에 0이 대입된다.
2. while지령다음에 test(중괄호)지령이 놓인다. num의 값이 10보다 작으면 순환이 진행된다.
3. 순환본체에서 num의 값이 하나씩 증가한다. num의 값이 전혀 변하지 않으면 순환은 무한히 또는 그 프로세스가 지우기될 때까지 반복된다.

실례 8-74

(The Script)

```
   #!bin/sh
   # Scriptname: quiz
1.  echo "Who was the chief defense lawyer in the OJ case?"
    read answer
2.  while [ "$answer" != "Johnny" ]
3.  do
        echo   "Wrong try again!"
4.      read answer
5.  done
6.  echo You got it!
```

(The Output)

$ **quiz**

Who was the chief defense lawyer in the OJ case? ***Marcia***
Wrong try agaun!
Who was the chief defense lawyer in the OJ case? ***I give up***
Wrong try again!
Who was the chief defense lawyer in the IJ case? ***Johnny***
You got it!

설명

1. echo지령은 사용자에게 who was the chief defense lamyer in the OJ case?를 현시한다. read지령이 사용자의 입력을 대기한다. 입력은 변수 answer에 기억된다.
2. while순환이 시작되고 test지령 즉 중괄호지령이 식을 검사한다. 변수 answer가 문자렬 Johnny와 같지 않으면 순환이 진행되여 do와 done사이의 지령들이 실행된다.
3. 예약어 do는 순환본체의 시작이다.
4. 사용자에게 입력할것을 요구한다.
5. 예약어 done은 순환본체의 끝을 현시한다. 조종이 while순환의 꼭대기로 올라 가고 식이 다시 검사된다. $answer가 Johnny와 같지 않는 한 순환이 계속 반복된다. 사용자가 Johnny를 입력하면 순환이 끝난다. 프로그람 조종이 행 6으로 넘어 간다.
6. 순환본체가 끝날 때 조종이 여기서 시작된다.

실례 8-75

(The Script)

$ **cat sayit**

```
   #'/bin/sh
   # Scripkname: sayit
```

```
    echo Type q to quit.
    go=start
1.  while [ -n "$go" ]             # Make sure to double quote the variable
    do
2.    echo -n I love you.
3.    read word
4.    if [ "$word" = q -o "$word" = Q]
      then
            echo "I'll always love you!"
            go=
      fi
    done

(The Output)
$ sayit
Type q to quit.
I love you.      When user presses the Enter key, the program continues
I love you.
I love you.
I love you.
I love you.q
I'll always love you!
$
```

설명

1. while다음의 지령이 실행되고 그의 탈퇴상태가 검사된다. test지령에 -n추가선택을 추가하면 빈값이 아닌 문자렬을 검사한다. go가 초기에 값을 가지므로 검사가 성공하고 탈퇴상태를 령으로 한다. 변수 go가 2중인용부호속에 있지 않고 빈값이면 test지령은 통보를 보낸다.

   go:test:argument expected

2. 순환이 시작된다. 문자렬 I love you가 현시장치에 출력된다.
3. read지령이 사용자입력을 대기한다.
4. 식이 검사된다. 사용자가 q나 Q를 입력하면 문자렬 I'll always love you!가 현시되고 변수 go에 빈값이 설정된다. 순환이 다시 시작될 때 변수가 빈값이므로 시험은 실패한다. 순환이 끝나고 조종이 done명령의 다음행으로 넘어 간다. 이 실례에서 스크립트는 실행할 행이 더 없으므로 끝난다.

until지령 until지령은 while지령과 같이 사용되지만 until다음의 지령이 실패할 때 즉 그 지령이 령이 아닌 탈퇴상태를 되돌릴 때에만 순환명령들을 실행한다. 예약어 done이 나타나면 조종이 순환의 꼭대기로 되돌아 가고 until지령이 지령의 탈퇴상태를 다시 검사한다. until에 의해 평가되는 지령의 탈퇴상태가 령으로 될 때까지 순환이 계속된다. 탈퇴상태가 령으로 되면 순환이 멎고 예약어 done다음부터 프로그람이 실행된다.

형식

```
until 지령
do
      지령(들)
done
```

실례 8-76

```
#!/bin/sh
1.  until who | grep linda
2.  do
        sleep 5
3.  done
    talk linda@dragonwings
```

설명

1. until순환은 파이프행에 있는 마지막지령인 grep의 탈퇴상태를 검사한다. who지령은 콤퓨터에 누가 등록가입되는가를 보여 주며 그 출력을 grep에 파이프한다. grep지령은 그것이 사용자 linda를 찾을 때만 탈퇴상태를 령(성공)으로 넘긴다.
2. 사용자 linda가 등록가입되지 않았으면 순환이 시작되여 프로그람이 5s동안 중지된다.
3. linda가 등록가입할 때 grep지령의 탈퇴상태가 령으로 되고 조종이 예약어 done 다음명령으로 넘어 간다.

실례 8-77

```
(The Script)
$ cat hour
      #!/bin/sh
      # Scriptname: hour
1.  hour=l
2.  until [ $hour -gt 24 ]
    do

3.   case “$hour” in
     [0-9] | 1[0-1])echo   “Good morning!”
               ;;
     12) echo   “Lunch time.”

               ;;
     1[3-7])echo   “Siesta time.”

              ;;
        *)   echo   “Good night.*
                ;;
       esac
```

4. **hours'expr $hour + 1***
5. **done**

(The Output)
$ **hour**
Good morning!
Good morning!
...
Lunch time.
Siesta time.
...
Good night.
...

설명

1. 변수 hour가 1로 초기화된다.
2. test지령은 hour가 24보다 큰가를 검사한다. hour가 24보다 크지 않으면 순환이 실행된다. until순환은 그다음의 지령이 령이 아닌 탈퇴상태를 넘기면 실행된다. 조건이 참일 때까지 순환이 계속 반복된다.
3. case지령은 hour변수를 평가하고 매 case명령들을 검사하여 맞는것이 있는가를 본다.
4. 조종이 순환의 꼭대기로 올라 가기전에 hour변수가 증가된다.
5. done지령은 순환본체의 끝을 표시한다.

순환지령 어떤 조건이 생기면 순환에서 벗어 나거나 순환의 꼭대기로 되돌아 가거나 또는 무한순환을 정지시킬 필요가 있을수 있다. Bourne쉘은 이러한 경우에 사용할수 있는 순환조종지령들을 제공한다.

shift지령 shift지령은 파라메터목록을 지적된 회수만큼 왼쪽으로 자리옮김한다. 인수가 없는 shift지령은 파라메터목록을 왼쪽으로 한번 자리옮김한다. 일단 목록이 자리옮김되면 그 파라메터는 영원히 지우기된다. 보통 위치파라메터들의 목록을 반복할 때 while순환에서 shift지령을 사용한다.

형식

shift [n]

실례 8-78

(Without a Loop)
(The Script)
1. set joe mary torn sam
2. **shift**
3. echo $*
4. **set 'date'**
5. echo $*

6. **shift 5**
7. echo $*
8. **shift 2**

(The Output)

3. *mary torn sam*
5. *Fri Sep 9 10:00:12 PDT 2001*
7. *2001*
8. ***cannot shift***

설명

1. set지령은 위치파라메터들을 설정한다. $1에 Joe가, $2에는 mary가, $3에는 tom이, $4에는 sam이 대입된다. $*는 모든 파라메터들을 표시한다.
2. shift지령이 위치파라메터들을 왼쪽으로 민다. 즉 joe가 밀기된다.
3. 밀기한 다음에 파라메터목록이 인쇄된다.
4. set지령은 위치파라메터들을 UNIX의 date지령의 출력에 재설정한다.
5. 새로운 파라메터목록이 인쇄된다.
6. 이때 목록이 왼쪽으로 5번 밀기된다.
7. 새로운 파라메터목록이 인쇄된다.
8. 존재하는 파라메터의 개수이상으로 밀기하면 쉘이 표준오유로 통보를 내보낸다.

실례 8-79

(With a Loop)

(The Script)

```
$ cat doit
      #'!/bin/sh
      # Name: doit
      # Purpose: shift through command line argument
      # Usage doit [args]
1.  while [   $# -gt   0 ]
    do
2.      echo $*
3.      shift
4.  done
```

(The Command Line)

$ **doit a b c d e**

a b c d e

b c d e

c d e
d e
e

설명

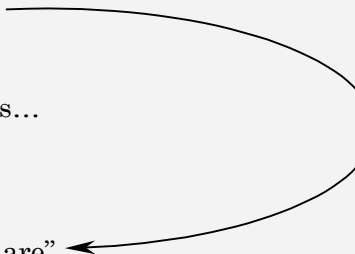
1. while지령은 수식을 검사한다. 위치파라메터들의 개수($#)가 0보다 크면 순환이 진행된다. 위치파라메터들은 지령행으로부터 인수로 넘어 온다. 5개의 위치파라메터가 있다.
2. 모든 위치파라메터들이 인쇄된다.
3. 파라메터목록이 한번 왼쪽으로 밀기된다.
4. 여기서 순환본체가 끝나고 조종이 순환꼭대기로 되돌아 간다. 순환이 시작될 때마다 shift지령은 파라메터목록이 한개씩 감소하게 한다. 첫번째 밀기후에 $#(위치파라메터의 개수)는 4이다. $#가 감소되여 령으로 될 때 순환이 끝난다.

실례 8-80

```
(The Script)
$ cat dater
      #!/bin/sh
      # Scriptname: dater
      # Purpose: set positional parameters with the set command
      # and shift through the parameters.
1.   set 'date'
2.   while [ $# -gt 0 ]
     do
3.       echo $1
4.       shift
     done

(The Output)
$ dater
Sat:
Oct
13
12:12:13
PDT
2001
```

설명

1. set지령은 date지령을 출력하여 위치파라메터 $1부터 $6까지에 대입한다.
2. while지령은 위치파라메터의 개수($#)가 0보다 큰가를 검사한다. 식이 참이면 순환이 진행된다.

3. echo지령은 첫번째 위치파라메터 $1의 값을 현시한다.
4. shift지령은 파라메터목록을 왼쪽으로 한번 밀기한다. 순환할 때마다 목록이 빌 때까지 밀린다. 목록이 빌 때 $#는 령으로 되고 순환이 끝난다.

break지령 내부지령인 break지령은 프로그람으로부터가 아니라 순환으로부터 즉시에 벗어 나도록 하는데 사용된다(프로그람에서 탈퇴하려면 exit지령을 사용한다.). break지령이 실행된후 예약어 done다음부터 조종이 시작된다. break지령은 제일 안쪽 순환으로부터 탈퇴하도록 한다. 따라서 순환고리들을 겹쌌다면 수를 인수로 하여 보다 밖의 지정된 순환으로부터 벗어 나도록 할수 있다. 만일 3개의 순환에 겹싸 졌다면 제일 바깥쪽 순환이 순환 3이고 그다음 겹쌓인 순환이 순환 2이며 제일 안쪽 겹쌓인 순환이 순환 1이다. break는 무한순환으로부터 탈퇴하는데 쓸모가 있다.

형식

```
break [n]
```

실례 8-81

```
1.  while true; do
2.   echo Are you ready to move on\ ?
     read answer
3.       if [  "$answer" = Y -o "$answer" = y ]
       then
4.           break
5.   else
       ....commands...
       fi
6.  done
7.  print "Here we are"
```

설명

1. true지령은 언제나 령상태로 탈퇴하는 UNIX지령이다. 무한순환을 시작할 때 자주 사용된다. 반두점으로 지령들을 분리시킬 때 while지령과 같은 행에 do명령문을 놓는것이 좋다. 순환이 진행된다.
2. 사용자입력이 요구된다. 사용자입력이 변수 answer에 대입된다.
3. $answer가 Y나 y와 같으면 조종이 4행으로 넘어 간다.
4. break지령이 실행되여 순환이 정지되고 조종이 7행으로 넘어 간다. 행 Here we are가 인쇄된다. 사용자가 Y나 y로 응답할 때까지 프로그람은 계속 입력을 요구한다. 이것은 영원히 계속 될수 있다.!
5. 3행에서 검사가 실패하면 else지령이 실행된다. 순환본체가 예약어 done에서 끝나고 조종이 다시 행 1에 있는 while의 짝대기에서 시작된다.
6. 이것이 순환본체의 끝이다.

7. break지령이 실행된후 여기서 조종이 시작된다.

continue지령 continue지령은 어떤 조건이 참으로 되면 조종을 순환의 꼭대기로 넘긴다. continue다음의 모든 지령들이 무시된다. 만일 많은 순환속에 겹싸 져 있다면 continue지령은 조종을 제일 안쪽에 있는 순환에 넘긴다. 수가 그의 인수로 주어 지면 조종은 임의의 순환의 꼭대기에서 시작될수 있다. 만일 3개의 순환속에 겹싸져 있다면 제일 바깥쪽 순환은 순환 3, 그다음 겹쌓인 순환은 순환 2, 제일 안쪽 겹쌓인 순환은 순환 1이다.[7]

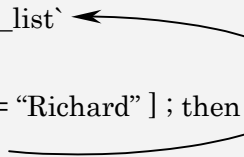
형식

```
continue [n]
```

실례 8-82

(The mailing List)

$ **cat mail_list**

ernie
john
richard
melaine
greg
robin

(The Script)

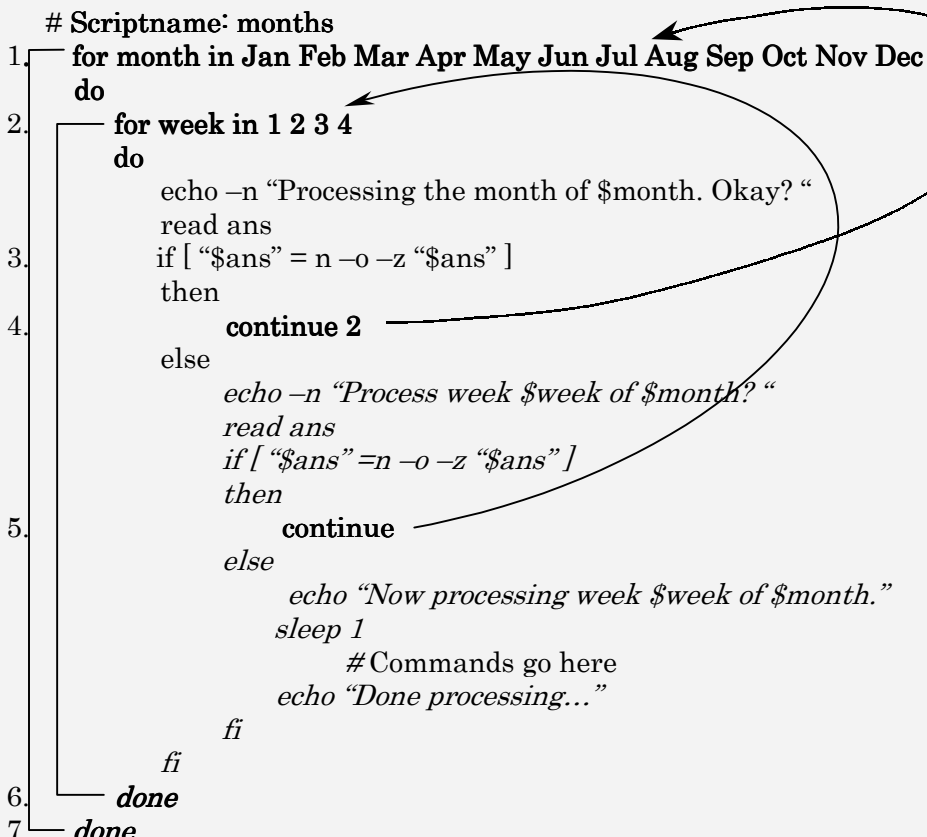
```
    #!/bin/sh
    # Scriptname: mailem
    # Purpose: To send a list
1.  for name in `cat mail_list`
    do
2.      if [    "$name" = "Richard" ] ; then
3.          continue
        else
4.          mail $name < memo
        fi
5.  done
```

설명

1. 지령바꿔넣기 cat mail_list후에 for순환은 mail_list라고 부르는 파일로부터 이름들의 목록을 반복한다.
2. 이름이 richard이면 continue지령이 실행되여 순환식이 평가되는 순환의

---

7. 만일 continue 지령에 순환의 개수보다 더 큰 수가 주어 지면 순환은 정지된다.

꽉대기로 조종이 다시 넘어 간다. richard가 이미 목록에서 밀기되였으므로 다음사용자 melanie가 변수 name에 대입된다. 여기서 문자렬 richard는 한개 단어이므로 인용부호안에 넣을 필요가 없다. 그러나 문자렬이 한개이상의 단어 실례로 richard jones로 이루어 졌다면 test지령이 오유통보 test:unknown operator richard를 내보내기때문에 test의 =연산자다음에 오는 문자렬을 인용부호안에 넣는것이 좋다.

3. continue지령은 순환본체의 임의의 나머지 지령들을 뛰여 넘어 조종을 순환의 꼭대기로 넘긴다.
4. richard를 제외한 목록의 모든 사용자들에게 파일 memo가 전달된다.
5. 이것이 순환본체의 끝이다.

겹쌓인 순환과 순환조종 겹쌓인 순환을 사용할 때 break나 continue지령들에 옹근수인수를 주어서 조종이 내부순환에서 바깥순환으로 넘어 가게 할수 있다.

실례 8-83

```
(The Script)
$ cat months
    # !/bin/sh
    # Scriptname: months
1.  for month in Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
    do
2.     for week in 1 2 3 4
       do
           echo –n "Processing the month of $month. Okay? "
           read ans
3.         if [ "$ans" = n –o –z "$ans" ]
           then
4.                 continue 2
           else
                 echo –n "Process week $week of $month? "
                 read ans
                 if [ "$ans" =n –o –z "$ans" ]
                 then
5.                     continue
                 else
                       echo "Now processing week $week of $month."
                     sleep 1
                           #Commands go here
                     echo "Done processing..."
                 fi
           fi
6.     done
7.  done
```

(The Output)
$ **months**
Processing the month of Jan. Okay?
Processing the month of Feb. Okay? ***y***
Process week 1 of Feb? ***y***
Now processing week 1 of Feb.
Done processing...
Processing the month of Feb. Okay? ***y***
Process week 2 of Feb? ***y***
Now processing week 2 of Feb.
Done processing...
Processing the month of Feb. Okay? ***n***
Processing the month of Mar. Okay? ***n***
Processing the month of Apr. Okay? ***n***
Processing the month of May. Okay? ***n***

설명

1. 바깥for순환이 시작된다. 첫번째 순환에서 Jan이 month에 대입된다.
2. 내부for순환이 시작된다. 첫번째 순환시 1이 week에 대입된다. 바깥순환으로 다시 넘어 가기전까지 내부순환이 계속 반복된다.
3. 사용자가 n을 입력하거나 Enter건을 누르면 이 행이 실행된다.
4. 인수가 2인 continue지령이 두번째 바깥순환의 꼭대기에서 조종을 시작한다. 인수가 없는 continue는 조종을 제일 안쪽 순환의 꼭대기로 넘긴다.
5. 조종이 제일 안쪽의 for순환으로 넘어 간다.
6. 이 done이 제일 안쪽 순환을 끝낸다.
7. 이 done이 제일 바깥순환을 끝낸다.

입출력방향바꾸기와 보조쉘 입력은 파일로부터 순환으로 파이프되거나 방향바꾸기될수 있다. 출력도 순환으로부터 파일로 파이프되거나 방향바꾸기될수 있다. 쉘은 보조쉘을 파생하여 입출력방향바꾸기를 하거나 파이프를 처리한다.

순환내부에서 정의된 임의의 변수는 순환이 끝날 때 스크립트의 나머지부분에 알려지지 않는다.

순환출력을 파일에 방향바꾸기

실례 8-84

(The Command Line)
1. $ **cat memo**
   abc
   def
   ghi

(The Script)
$ **cat numbers**
```
#!/bin/sh
```

```
    # Program name: number 11
    # Put line numbers on all lines of memo
2.  if [ $# -lt   1 ]
    then
3.      echo   "Usage: $0 filename " >&2
        exit 1
     fi
4.  count=l                              # Initialize count
5.  cat $1 | while read line
    # Input is coming from file on comma.nd line
    do
6.      [ $count -eq 1 ] && echo "Processing file $!..." > /dev/tty
7.      echo $count $line
8.      count='expr $count + 1'
9.  done > tmp$$                         # Output is going to a temporary file
10. mv tmp$$ $1

(The Command Line)
11. $ numberit   memo
    Processing file memo...

12. $ cat mmemo
    1 abc
    2 def
    3 ghi
```

설명

1. 파일 memo의 내용이 현시된다.
2. 이 스크립트를 실행할 때 사용자가 지령행인수를 제공하지 않았으면 인수들의 개수($#)는 1보다 작고 오유통보가 나타난다.
3. 인수들의 개수가 1보다 작으면 사용통보가 표준오유(>&2)로 전송된다.
4. count변수에 값 1이 대입된다.
5. UNIX의 cat지령이 $1에 기억된 파일이름의 내용들을 현시하고 출력이 while순환에 파이프된다. read지령의 첫번째 순환에서 파일의 첫번째 행이 대입되고 다음번 순환에서 두번째 행이 대입되는 식으로 계속된다. read지령은 입력을 읽는데서 성공하면 탈퇴상태를 령으로, 실패하면 1로 넘긴다.
6. count의 값이 1이면 echo지령이 실행되여 그 출력이 현시장치 /dev/tty에 전송된다.
7. echo지령은 count의 값을 인쇄하고 그뒤에 파일에 있는 행을 인쇄한다.
8. count가 한개씩 증가한다.
9. 전체 순환의 출력인 $에 있는 파일의 매 행이 말단 /dev/tty로 방향바꾸기되는 파일의 첫번째 행을 제외하고 파일 tmp$$에 방향바꾸기된다.[8]
10. tmp파일의 이름을 $1에 대입된 파일이름으로 다시 짓는다.
11. 프로그람이 실행된다. 처리되는 파일을 memo라고 부른다.

---

8. $$는 현재쉘의 PID 번호로 확장된다. 파일이름에 이 번호를 붙여서 파일이름을 구별한다.

12. 파일 memo가 스크립트가 끝난 다음 현시되여 행번호들이 매 행의 앞에 첨부되였다는것을 보여 준다.

실례 8-85

```
(The File)
$ cat testing
apples
pears
peaches

(The Script)
    #!/bin/sh
    # This program demonstrates the scope of variables when
    # assigned within loops where the looping command uses
    # redirection. A subshell is started, when the loop uses
    # redirection, making all variables created within the loop
    # local to the shell where the loop is being executed.
1.  while read line
    do
2.      echo $line          # This line will be redirected to outfile
3.      names JOE
4.  done < testing > outfile    # Redirection of input and output
5.  echo Hi there $name

(The Output)
5.  Hi there
```

설명

1. read지령의 탈퇴상태가 성공이면 while순환이 시작된다. read지령은 4행의 done다음에 있는 파일 testing으로부터 입력을 받는다. 순환할 때마다 매번 read지령은 파일 testing으로부터 또 다른 행을 읽는다.
2. line의 값이 4행에 있는 outfile에 방향바꾸기된다.
3. 변수 name에 JOE가 대입된다. 이 순환에서 방향바꾸기가 사용되므로 변수는 그 순환에 대하여 국부변수로 된다.
4. 예약어 done은 파일 testing으로부터의 입력방향바꾸기와 파일 outfile에로의 출력방향바꾸기로 이루어 진다. 이 순환의 모든 출력은 outfile로 나간다.
5. name은 순환밖에 있을 때 정의되지 않는다. 그것은 while순환에 국부적이고 그 순환본체내에서만 알려 져 있다. 변수 name이 값을 가지지 않으므로 문자렬 Hi there만이 현시된다.

순환출력을 UNIX지령에 파이프하기 출력은 또 다른 지령에 파이프되거나 파일에 방향바꾸기될수 있다.

실례 8-86

```
(The Script)
     #!/bin/sh
1.  for i in 7 9 2 3 4 5
2.  do
      echo $i
3.  done | sort -n

(The Output)
2
3
4
5
7
9
```

설명

1. for순환이 분류되지 않은 수들의 목록을 반복한다.
2. 순환본체에서 수들이 인쇄된다. 이 출력은 수분류지령인 UNIX sort지령에 파이프된다.
3. 예약어 done다음에 파이프가 작성된다. 보조쉘에서 순환이 실행된다.

배경에서 순환실행 순환은 배경에서 실행시켜 진행할수 있다. 프로그람은 순환이 프로쩨스를 끝내는것을 기다리지 않고 계속 실행할수 있다.

실례 8-87

```
(The Script)
#!/bin/sh
1.  for person in bob jim joe sam
    do
2.        mail $person < memo
3.  done &
```

설명

1. for순환은 단어목록에 있는 매 이름들 즉 bob, jim, joe, sam을 밀기한다. 매 이름들의 차례로 변수 person에 대입된다.
2. 순환본체에서 person에 memo파일의 내용이 보내진다.
3. 예약어 done의 끝에 있는 앰퍼샌드(&)는 순환이 배경에서 실행되도록 한다. 프로그람은 순환이 진행될 때 계속 실행된다.

exec지령과 순환 exec지령을 사용하여 보조쉘을 작성함이 없이 표준입력과 표준출력을 열거나 닫을수 있다. 따라서 순환을 시작할 때 순환본체에서 작성된 임의의 변수들은 순환이 끝날 때 그대로 보존된다. 순환에서 방향바꾸기를 사용할 때 순환안에서 작성된 임의의 변수들은 없어 진다.

보통 exec지령을 사용하여 읽기와 쓰기를 위해 이름이나 파일서술자번호로 파일들

을 연다. 파일서술자 0, 1, 2는 각각 표준입력, 표준출력, 표준오유에 대응하여 예약되여 있다. 파일이 열려 지면 그것은 다음번 사용할수 있는 파일서술자를 가진다. 실례로 파일서술자 3이 다음번 빈서술자이면 새로운 파일에 파일서술자 3이 대입된다.

실례 8-88

```
(The File)
1.  $ cat tmp
    apples
    pears
    bananas
    pleaches
    plums

(The Script)
$ cat speller
    #!/bin/sh
    # Scriptname: speller
    # Purpose: Check and fix spelling   errors in a file
2.  exec < Imp                          # Opens the tmp file
3.  while read line                     # Read from the tmp file
    do
4.      echo $line
5.      echo  -n  "Is this word correct? [Y/N] "
6.      read answer < /dev/tty       # Read from the terminal
7.      case  "$answer" in
8.      [Yy]*)
9.             continue; ;
      *)
             echo  "What is the correct spelling? "
10.          read word < /dev/tty
11.          sed  "s/$line/$word/g" tmp > error
12.          mv error tmp
13.          echo $line has been changed to $word.
      esac
14. done
```

설명

1. tmp파일의 내용들이 현시된다.
2. exec지령이 표준입력(파일서술자 0)을 변경시켜 입력이 건반이 아니라 tmp 파일로부터 들어 오게 한다.
3. while순환이 시작된다. read지령은 tmp파일로부터 입력행을 받는다.
4. line변수에 기억된 값이 인쇄된다.
5. 단어가 정확한가를 사용자에게 질문한다.
6. read는 말단 /dev/tty로부터 사용자의 응답을 받는다. 만일 입력이 직접 말단으로부터 방향바꾸기되지 않는다면 읽기 위해 열려 져 있는 파일 tmp로부터 읽기를 계속한다.

7. case지령은 사용자의 대답을 평가한다.
8. 변수 answer가 Y나 y로 시작되는 문자렬과 같다면 다음행에 있는 continue명령이 실행된다.
9. continue명령은 프로그람이 3행에 있는 while순환의 시작으로 넘어 가도록 한다.
10. 사용자에게 다시 입력(맞춤법이 정확한 단어)이 요구된다. 입력이 말단 /dev/tty로부터 방향바꾸기된다.
11. sed지령은 line의 값이 tmp파일에서 나타나는 곳마다에서 그 값을 word의 값으로 바꾸고 그 출력을 error파일에 보낸다.
12. error파일은 이름이 고쳐 진 tmp파일이므로 tmp의 낡은 내용들우에 error파일의 내용들이 씌여 진다.
13. 이 행이 현시되여 변화가 생겼다는것을 알려 준다.
14. 예약어 done이 순환본체의 끝을 현시한다.

IFS와 순환 쉘의 내부마당분리기호(IFS)는 공백, 타브 그리고 행바꾸기이다. 이것은 read, set 그리고 for와 같이 단어들의 목록을 해석하는 지령들을 위해 단어(통표)분리기호로 사용된다. 만일 한 목록에서 다른 분리기호가 사용되면 그것이 사용자에 의해 재설정될수 있다. 그 값을 변화시키기전에 IFS의 초기값을 다른 변수에 보존하는것이 좋다. 그러면 필요할 때 그의 기정값으로 되돌아 가기가 쉽다.

실례 8-89

```
(The Script )
$ cat runit
#/bin/sh
# Script is called runit.
# IFS is the internal field sepwscor and defaults to
# spaces, tabs, and newlines.
# In this script it is changed to a colon.
1.  names=Tom:Dick:Harrys John
2.  OLDIFS="$IFS"      # Save the orginal value of IFS
3.  IFS=":"
4.  for persons in $names
     do
5.       echo Hi $persons
        done
6.  IPS="$OLDIFS"                # Reset the IFS to old value
7.  set Jill Jane Jolene           # Set positional parameters
8.  for girl in $*
     do
9.       echo Howdy $girl
     done
(The Output)
$ runit
5   Hi Tom
```

Hi Dick
Hi Harry
Hi John
9 Howdy Jill

Howdy Jane
Howdy Jolene

설명

1. 변수 names에 문자렬 Tom: Dick:Harry:John이 설정된다. 매 단어들은 옹근두점에 의해 분리된다.
2. IFS의 값인 공백이 다른 변수 OLDIFS에 대입된다. IFS의 값은 공백이므로 그것을 보존하기 위해 인용부호안에 넣어야 한다.
3. IFS에 옹근두점이 대입된다. 옹근두점은 단어들을 분리시키는데 사용된다.
4. 변수바꿔넣기한후 for순환은 옹근두점을 단어들사이의 내부마당분리기로 사용하여 매 단어들을 반복한다.
5. 단어목록에 있는 매 단어들이 현시된다.
6. IFS에 OLDIFS에 기억된 초기값이 다시 대입된다.
7. 위치파라메터들이 설정된다. $1에 Jill이, $2에 Jane이 그리고 $3에 Jolene가 대입된다.
8. $*는 모든 위치파라메터들 Jill, Jane, Jolene을 현시한다. for순환은 순환을 반복할 때마다 매 이름들을 차례로 girl변수에 대입한다.
9. 파라메터목록에 있는 매 이름들이 현시된다.

8. 함수

함수들은 AT&T의 UNIX체계 VR2에서 Bourne쉘에 소개되였다.

함수는 한개 지령이나 지령들의 묶음에 대한 이름이다. 함수들을 사용하여 프로그람을 모듈화하며 더 효률적으로 작성할수 있다. 사용자들은 함수들을 다른 파일에 기억시키고 그것을 사용하려고 할 때 스크립트에 적재할수 있다.

함수사용에 대한 중요한 규칙들을 보면 다음과 같다.

1. Bourne쉘은 사용자가 내부지령, 함수 또는 디스크에 있는 실행프로그람을 사용하고 있는가를 결정한다. 그것은 먼저 내부지령을 찾고 다음에 함수, 마지막에 실행프로그람을 찾는다.
2. 함수는 사용하기전에 정의되여야 한다.
3. 함수는 현재환경에서 실행된다. 함수는 그것을 호출하는 스크립트와 변수들을 공유하고 인수들을 위치파라메터들로 대입하여 넘긴다. 만일 함수에서 exit지령을 사용하면 전체 스크립트에서 탈퇴한다. 그러나 함수의 입력과 출력이 방향바꾸기되거나 함수가 거꿀인용부호(지령바꿔넣기)속에 닫겨 있으면 보조쉘이 작성되고 그 함수와 변수들 그리고 현재작업등록부가 보조쉘내에서만 알려진다. 함수가 탈퇴할 때 거기서 설정된 변수들은 없어 지고 만일 등록부를 변경시켰다면 함수를 호출하기전에 있었던 등록부로 되돌아 간다. 함수에서 탈퇴하면 그 함수가 호출되였을 때 스크립트가 벗어 났던 곳으로 되돌아 간다.

4. return명령은 함수내에서 실행된 마지막지령의 탈퇴상태와 주어 진 인수의 값을 되돌리는데 그 값은 255를 초과할수 없다.
5. 함수들은 그것들이 정의된 쉘에서만 존재한다. 그것들은 보조쉘로 넘겨 질수 없다. 점지령을 사용하여 파일에 기억된 함수들을 실행할수 있다.
6. 함수들과 정의들을 현시하기 위하여 set지령을 사용한다.
7. 트랩프는 변수와 같이 함수내에서 전역적이다. 그것들은 스크립트와 거기서 호출되는 함수들에 의해 공유된다. 만일 트랩프가 함수내에서 정의되면 스크립트에 의해서도 공유된다. 이렇게 하면 바라지 않는 효과가 생길수 있다.
8. 함수들이 다른 파일에 기억되면 점지령으로 그것들을 현재스크립트에 적재할 수 있다.

형식

function-name(){ 지령들 ; 지령들;}

실례 8-90

dir () {echo “directories:”; ls-l | nawk'/d/{print $nf}';}

설명

함수의 이름은 dir이다. 빈괄호는 함수를 이름 짓는 문법이지 다른 목적은 없다. dir가 입력되면 대괄호안에 있는 지령들이 실행된다. 이 함수의 목적은 현재작업등록부밑에 있는 부분등록부들만을 현시하자는것이다. 대괄호옆에 공백이 있어야 한다.

함수의 설정해제 기억기에서 함수를 지우기하기 위하여 unset지령을 사용한다.

형식

unset 함수이름

함수인수와 되돌이값 함수가 현재쉘에서 실행되므로 변수들은 함수와 쉘에 다 알려진다. 함수에서 환경이 변화되면 쉘에서도 변화된다. 인수들은 위치파라메터들을 사용하여 함수에 넘길수 있다. 위치파라메터들은 그 함수에서만 쓸수 있다. 즉 함수에 대한 인수들은 함수밖에서 사용되는 임의의 위치파라메터들에 영향을 주지 않는다.

return지령을 사용하여 함수에서 벗어 난 조종을 그 함수가 호출되였던 곳에서 프로그람에 다시 넘길수 있다(만일 함수내부와 스크립트의 임의의 곳에서 exit를 사용하면 스크립트가 끝난다는것을 명심하여야 한다.). 함수의 되돌이값은 return지령에 어떤 인수를 주지 않는 한 스크립트에서 마지막지령의 탈퇴상태의 값과 같다. return지령에 값을 대입하면 그 값은 ?변수에 기억되는데 0부터 255사이의 옹근수값을 가질수 있다. return지령은 0과 255사이의 옹근수값만을 되돌리게 제한되므로 지령을 대입하여 함수의 출력을 얻을수 있다. 만일 UNIX지령의 출력을 얻으려면 전체 함수를 거꿀인용부호속에 넣고 그 출력을 변수에 대입하여야 한다.

실례 8-91

(Using the return Conimand)
(The Script)

```
$ cat do_increment
    #!/bin/sh
    # Scriptname: do_increment
1.  increment () {
2.      sum='expr $1+1'
3.      return $sum   # Return the value of sum to the script.
    }
4.  echo -n "The sum is "
5.  increment 5       # Call function increment; pass 5 as a
                      # parameter. 5 becomes $1 for the increment
                      # function.
6.  echo $?           # The return value is stored in $?
7.  echo $sum         # The variable "susa" is known to the function,
                      # and is also known to the main script.

(The Output)
     $ do_increment
4, 6  The sum is 6
7     6
```

설명

1. increment라고 하는 함수가 정의된다.
2. 함수가 호출될 때 첫번째 인수 $1의 값이 1만큼 증가되여 그 결과가 sum에 대입된다.
3. 내부지령 return은 인수가 주어 지면 함수가 호출된 행다음의 기본스크립트로 복귀된다. 그것은 자기 인수를 ?변수에 기억한다.
4. 문자렬이 현시장치에 출력된다.
5. increment함수는 인수를 5로 하여 호출된다.
6. 함수가 복귀될 때 그 탈퇴상태가 ?변수에 기억된다. 탈퇴상태는 지적된 인수가 return명령에 사용되지 않는 한 함수에서 제일 마지막에 실행되는 지령의 출력값이다. return의 인수는 반드시 0으로부터 255사이의 옹근수가 되여야 한다.
7. sum이 함수 increment에서 정의되였지만 그 유효범위가 전역적이므로 그 함수를 호출한 스크립트에서도 알려 진다. 그 값이 인쇄된다.

실례 8-92

```
(Using Command Substitution)
(The Script)
$ cat do_square
    #!/bin/sh
    # Scriptname: do_square
1.  function square {
    sq='expr $1 \ * $1'
```

```
     echo "Number to be scuared is $1."
2.   echo "The result is
     }
3.   echo "Give me a number to square. "
     read number
4.   value_returned=` square $number `   # Command substitution
5.   echo    $value_returned
```

(The Output)

```
$  do_square
3.  Give me a number to square.
    10
5.  Number to be squared is 10. The result is 100
```

설명

1. square라고 부르는 함수가 정의된다. 그 기능은 호출될 때 인수 $1에 자기 자체를 곱하는것이다.
2. 수를 두제곱한 결과가 인쇄된다.
3. 사용자입력이 요구된다. 이 행에서 프로그람이 실행을 시작한다.
4. 함수 square가 수(사용자 입력)를 인수로 받아 호출된다. 함수가 거꿀인용부호속에 있기때문에 지령바꿔넣기가 진행된다. 함수의 출력, 2개의 echo지령의 출력이 모두 변수 value_returned에 대입된다.
5. 지령을 대입했으므로 문자렬 Number to be squared is 10과 The result is 100사이의 행바꾸기가 없어 진다.

9. 함수와 dot지령

함수기억 함수들은 보통 .porofile파일에 정의되므로 사용자가 등록가입할 때 정의된다. 함수들은 넘겨 질수 없지만 파일에 기억될수 있다. 따라서 함수가 필요할 때에는 그 파일의 이름과 함께 dot지령을 사용하여 그안에서 함수들이 정의된것처럼 한다.

실례 8-93

```
1.  $ cat myfunctions
2.  go( ) {
      cd $HOME/bin/prog
      PS1 = ' 'pwd' > '
      ls
    }
3.  greetings() { echo "Hi $1! Welcome to my world." ; }
4.  $ . myfunctions
5.  $ greetings george
Hi geroge! Welcome to my world.
```

설명

1. 파일 myfunctions가 현시된다. 거기에 2개의 함수정의가 있다.
2. 첫번째 정의된 함수는 go라고 부른다. 그것은 1차재촉문을 현재작업등록부로 설정한다.
3. 두번째 정의된 함수는 greetings라고 한다. 그것은 인수로서 넘겨 진 사용자이름을 받는다.
4. dot지령이 파일 myfunctions의 내용들을 쉘의 기억기에 적재한다. 이때 2개의 함수가 쉘에서 정의된다.
5. greetings함수가 호출되여 실행된다.

실례 8-94

(The *.dbfunctions* file shown below contain functions to be used by the main program)

```
1.  $ cat .dbfunctions
2.  addon () {   # Function is named and defined in    file .dbfunctions
3.      while true
        do
            echo “Adding information “
            echo “Type the full name of employee “
            read name
            echo “Type address for employee “
            read address
            echo “Type start date for employee (4/10/88 ) ;”
            read startdate
            echo $name:$address:$startdate
            echo –n “Is this correct? “
            read ans
            case “$ans”   in
            [Yy]*)
                        echo “Adding info…”
                        echo $name:$address:$startdate>>datafile
                sort –u datafile –o datafile
                echo –n “Do you want to go back to the main menu? “
                read ans
                if [$ans = Y –o $ans = y ]
                then
4.                      return          # Return to calling program
                else
5.                      continue      # Go to top of the loop
                fi
                ;;
            *)
                echo “Do you want to try again? “
                read answer
```

```
                case "$answer" in
                [Yy]*) continue;;
                *) exit;;
                esac
                     ;;
             esac
        done
6.   }    # End of function definition
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(The Command Line)
7.   $ more mainprog
     #!/bin/sh
     # Scriptname: main script that will call the function, addon

     datafile=$HOME/bourne/datafile
8    . .dbfunctions        # The dot command reads the dbfunctions file
                              # into memory
    if [ ! –f $datafile ]
    then
       echo "  `basename $datafile  `  dose not exist" 1>&2
       exit 1
  fi
  9.   echo "Select one: "
      cat << EOF
      [1] Add info
      [2] Delete info
      [3] Exit
          EOF
          Read chioce
          Case $chioce in
 10.        1)   addon             # Calling the addon function
                    ;;
            2)   delete            # Calling the delete function
                    ;;
            3)   update
                    ;;
            4)
                 echo Bye
                 exit   0
                    ;;
            *)   echo Bad chioce
                 exit   2
                    ;;
     esac
     echo Returned from function call
     echo The name is $name
   # Variable set in the function are known in this shell.
```

설명

1. .dbfunctions파일이 현시된다.
2. addon함수가 정의된다. 이 기능은 파일 datafile에 새 정보를 추가하는것이다.
3. while순환이 시작된다. break나 continue 같은 순환조종명령이 순환본체에 포함되여 있지 않으면 순환이 영원히 계속된다.
4. return지령이 함수가 호출되였던 호출프로그람으로 조종을 다시 넘긴다.
5. 조종이 while순환의 꼭대기로 다시 올라 간다.
6. 닫는 대괄호는 함수정의를 끝낸다.
7. 이것이 기본스크립트이다. 이 스크립트에서 함수 addon이 사용된다.
8. dot지령은 파일 .dbfunctions를 프로그람의 기억기에 적재한다. 이때 함수 addon은 스크립트에 대해 정의되고 사용할수 있게 된다. 그것은 마치도 스크립트의 바로 이 부분에서 함수를 정의한것이나 같다.
9. here문서와 함께 차림표가 현시된다. 사용자는 차림표항목을 선택할것을 요구한다.
10. addon함수가 호출된다.

10. 신호트랩프

프로그람이 실행될 때 Control-C 나 Control-\ 를 누르면 그 신호가 도착하자마자 프로그람이 중지된다. 신호가 도착하자마자 즉시에 프로그람이 중지되지 않도록 할 필요가 있을수도 있다. 이때 신호를 무시하고 프로그람을 계속 실행시키던가 또는 스크립트를 벗어 나기전에 어떠한 정돈조작을 진행하도록 조종할수 있다. trap지령은 프로그람이 신호를 받을 때 동작하는 방식을 조종한다.

신호는 한 프로쎄스에서 다른 프로쎄스에 전송되거나 또는 어떤 건반들이 눌리워 지거나 뜻밖의 일이 일어 났을 때 조작체계가 프로쎄스에 보낼수 있는 한개의 번호로 된 비동기통보이다. trap지령은 쉘이 신호를 받으면 현재 실행하고 있는 지령을 끝내라고 알려 준다. 만일 trap지령이 인용부호안에 있는 지령들앞에 놓이면 그 지령문자렬은 지정된 신호를 받은 상태에서 실행된다. 쉘은 지령문자렬을 두번 읽는데 한번은 트랩프가 설정될 때 읽고 다른 한번은 신호가 도착할 때 다시 읽는다. 지령문자렬이 2중인용부호안에 있다면 트랩프가 첫번째로 설정될 때 모든 변수와 지령바꿔넣기가 진행된다. 만일 지령문자렬이 단일인용부호안에 있으면 변수와 지령바꿔넣기는 신호가 검사되여 trap가 실행될 때까지 진행되지 않는다.

모든 신호들의 목록을 얻자면 지령 kill-l을 사용하여야 한다. 표 8-15는 신호번호와 그에 대응한 이름들의 목록을 보여 준다.

형식

trap' 지령; 지령' 신호번호

실례 8-95

trap' rm tmp*; exit ? 1 2 15

설명

신호 1(정지), 신호 2(새치기), 신호 15(프로그람 끝)중에서 어느 하나가 도착하면 모든 tmp파일들을 지우고 탈퇴한다. 스크립트가 실행될 때 새치기신호가 오면

trap지령은 이 신호를 여러가지 방법으로 처리하게 한다. 사용자는 신호가 표준대로 (기정으로)처리되도록 할수 있으며 또한 신호를 무시할수도 있고 신호가 도착할때 호출되는 프로세스함수를 작성할수 있다.

표 8-15. 신호번호와 신호

1)HUP	12)SYS	23)POLL
2)INT	13)PIPE	24)XCPU
3)QUIT	14)ALRM	25)XFSZ
4)ILL	15)TERM	26)VTALRM
5)TRAP	16)URG	27)PROF
6)IOT	17)STOP	28)WINCH
7)EMT	18)TSTP	29)LOST
8)FPE	19)CONT	30)USR1
9)KILL	20)LOST	31)USR2
10)BUS	21)TTIN	
11)SEGV	22)TTOU	

신호재설정 신호가 기정대로 처리되도록 재설정하기 위해 trap지령뒤에 신호이름이나 번호를 쓴다.

실례 8-96

```
trap 2
```

설명

프로세스를 중지시키는데 사용되는 신호 2인 SIGINT 즉 Control-C에 대한 기정동작을 재설정한다.

실례 8-97

```
trap 'trap 2' 2
```

설명

신호가 도착할 때 인용부호속에 있는 지령문자렬을 실행하는 신호2(SIGINT)에 대한 기정동작을 설정한다. 사용자는 프로그람을 끝내기 위하여 Contorl- C를 두번 눌러야 한다. 첫번째 trap는 신호를 접수하고 두번째 trap는 프로세스를 끝내는 기정동작으로 트랩프를 재설정한다.

신호무시 trap지령뒤에 한쌍의 빈인용부호가 놓이면 현시된 신호들이 프로세스에 의해 무시된다.

실례 8-98

trap “ “ 1 2

설명

신호 1과 2가 쉘에 의해 무시된다.

트랩프목록제시 모든 트랩프들과 그에 대입된 지령들의 목록을 제시하기 위해 trap 를 입력한다.

실례 8-99

```
(The Script)
$ cat trapping
      #/bin/sh
      # Scriptname: trapping
      # Script to illustrate the trap cpmmand and signals
1.  trap 'echo "Control-C will not terminate $0."' 2
2.  trap 'echo "Control-\   will not terminate $0."' 3
3.  echo "Enter any string after the prompt. "
    echo "When you are ready to exit, type\ ”stop\ "."
4.  while true
    do
         echo -n "Go ahead...-> ”
         read reply
5.       if [ "$reply" .= stop ]
         then
6.              break
         fi
7.  done

(The Output)
$   trapping
Enter any string after the prompt.
When you are ready to exit, type “stop”.
Go ahead...> this is it^C
Control-C will not terminate trapping
Go ahead...> this is it ^C
Control-\   will not terminate trapping.
Go ahead...> stop
$
```

설명

1. 첫번째 trap는 Control-C인 INT신호를 받는다. 만일 ^C가 프로그람이 실행될 때 눌리워 지면 인용부호속에 있는 지령이 실행된다. 프로그람은 실패하지 않고 Control-C will not terminate trapping을 인쇄하고 사용자가 입

력할것을 계속 재촉한다.
2. 사용자가 QUIT신호인 Control-\ 을 누를 때 두번째 trap지령이 실행된다. 문자렬 Control-\ will not terminate trapping 이 현시되고 프로그람이 계속 실행된다. 이 신호는 기정으로 프로쎄스를 끌내고 기본파일을 생성한다.
3. 사용자가 입력할것을 재촉한다.
4. while순환이 시작되고 재촉문 Go ahead …가 현시된다.
5. 사용자입력이 reply변수에 대입되고 그 값이 staph와 같으면 순환에서 벗어나 프로그람이 끝난다. 따라서 kill지령으로 프로그람이 중지되지 않으면 이 방법을 써서 프로그람에서 벗어 날수 있다.
6. break지령은 순환본체에서 벗어 나 조종이 7행다음부터 시작되도록 한다. 이 경우 프로그람이 여기서 끝난다.
7. 이것이 while순환의 끝이다.

함수에서 트랩프 만일 트랩프를 사용하여 함수에서 신호를 처리할 때 함수가 호출되면 그것은 전체 스크립트에 영향을 준다. 트랩프는 스크립트에 대하여 전역적이다.

다음의 실례에서 새치기건ˆC를 무시하기 위해 트랩프가 설정된다. 순환을 멈추기 위해 kill지령을 사용하여 스크립트를 정지시켜야 했다. 이 실례는 함수에서 트랩프를 사용할 때 생길수 있는필요 없는 부차적인 효과를 보여 준다.

실례 8-100

```
(The Script)
   #!/bin/sh
1.  traper ( ) {
                 echo "In trapper"
2.       trap   'echo "Caught in a trap!"' 2
         # Once set, this trap affects the entire script. Anytime
         # ^C is entered, the script will ignore it.
    }
3.  while :
    do
         echo "In the main script"
4.           trapper
5.           echo "Still in main"
             sleep 5
       done

(The Output)
$ trapper
In the main script
In trapper
Still in main
^Ccaught in a trap!
In the main script
```

In trapper
Still in main
^Ccaught in a trap!
In the main script

설명

1. trapper함수가 정의된다. 함수에서 설정된 모든 변수들과 트랩프들이 스크립트에 대하여 전역적이다.
2. trap지령은 새치기건(ˆC)신호 2를 무시한다. 만일 ˆC가 눌리워 지면 통보 Caught in a trap가 인쇄되고 스크립트는 영원히 계속 실행된다.
3. 기본스크립트가 무한순환을 시작한다.
4. 함수 trapper가 호출된다.
5. 함수가 복귀할 때 여기서 실행이 시작된다.

오유수정 sh지령에 -n추가선택을 사용하여 임의의 지령들을 실행시키지 않고도 스크립트의 문법을 검사할수 있다. 스크립트에 문법적오유가 있으면 쉘은 그 오유를 통보한다. 오유가 없으면 아무것도 현시되지 않는다.

스크립트를 오유수정하는데 가장 일반적으로 사용되는 방법은 -x추가선택과 함께 set지령을 사용하거나 -x추가선택을 sh지령에 대한 인수로서 스크립트이름앞에서 사용하는것이다. 오유수정추가선택들의 목록을 표 8-16에서 보여 주었다. 이 추가선택들은 스크립트의 실행을 추적할수 있게 한다. 대입이 진행된 다음에 스크립트의 매 지령이 현시되고 그다음에 지령이 실행된다. 스크립트로부터 행이 현시될 때 그앞에 더하기(+)부호가 놓인다.

verbose추가선택이 설정되면 즉 Bourne쉘을 -v추가선택(sh -v스크립트이름)으로 호출하면 스크립트의 매행이 스크립트에 입력된것과 꼭같이 현시되고 그다음에 실행된다.

표 8-16. 오유수정추가선택

지 령	추가선택	기 능
sh-x스크립트이름	echo추가선택	스크립트의 매행에 변수를 대입하고 실행하기전에 현시한다.
sh-v스크립트이름	verbose추가선택	스크립트의 매행을 실행전에 입력된것과 꼭같이 현시한다.
sh-n스크립트이름	noexec추가선택	지령을 해석하지만 실행하지 않는다.
set-x	echo를 선택한다.	스크립트에서 실행을 추적한다.
set+x	echo를 해제한다.	추적하지 않는다.

실례 8-101

(The Script)
$ **cat todebug**
#!/bin/sh
1. *# Sripttname todebug*

```
    name="Joe Blow"
    if  [ "$name" ="Joe Blow" ]
    then
          echo "Hi $name"
    fi
num=l
while [ $num -lt 5 ]
do
      num= ' expr $num + 1'
done
echo The grand total is $num

(The Output)
2   $ sh -x todebug
   + name=<Joe Blow
   + [ Joe Blow =   Joe Blow ]
   + echo Hi Joe Blow
Hi Joe Blow
   num=l
   + [ 1 -lt 5 ]
   + expr 1+1
   num=2
   + [ 2 -lt 5 ]
   + expr 2+1
   num=3
   + [ 3 -lt 5 ]
   + expr 3+1
   num=4
   + [ 4 -lt 5 ] + expr 4+1                 "*
   num=5
   + [ 5 -lt 5 ]
   + echo The grand total is 5 The grand total is 5
```

설명

1. 스크립트는 todebug라고 부른다. -x추가선택이 선택될 때 스크립트가 실행되는것을 볼수 있다. 순환의 매 반복이 현시되고 변수들의 값이 설정된대로 변화될 때마다 인쇄된다.
2. sh지령은 Bourne쉘을 -x추가선택로 기동한다. Echoing이 설정된다. 스크립트의 매행에 더하기부호가 앞에 붙어 현시장치에 현시된다. 행이 현시되기전에 변수바꿔넣기가 진행된다. 지령의 실행결과가 행이 현시된 다음에 나타난다.

11. getopts에 의한 지령행추가선택처리

몇개의 지령행추가선택들이 있는 스크립트들을 작성할 때 위치파라메터들이 제일 효과적인것은 아니다. 실례로 UNIX의 ls지령은 몇개의 지령행추가선택들과 인수들을

가진다(한개의 추가선택은 한개의 안내횡선을 요구한다. 인수는 요구하지 않는다.). 추가선택들은 여러가지 방법 즉 ls-laFi, ls-i-a-l-F, ls -ia-F등과 같이 넘겨 질수 있다. 만일 인수들을 요구하는 스크립트가 있다면 위치파라메터들을 사용하여 인수들을 ls-l-i-F와 같이 개별적으로 처리할수 있다. 매 횡선추가선택은 $1, $2, $3에 각각 기억된다. 그런데 사용자가 ls-liF에서와 같이 모든 추가선택들을 하나의 횡선으로 표시했다면 어떻게 되겠는가 이때 -liF는 스크립트에서 모두 $1에 대입된다. getopts함수는 지령행추가선택들과 인수들을 ls프로그람에 의해서 처리되는것과 같은 방법으로 처리할수 있게 한다. getopts함수를 사용하면 runit프로그람이 인수들을 여러가지로 조합하여 처리할수 있다.

실례 8-102

(The Command Line)

1. $ **runit -x -n 200 filex**
2. $ **runit -xn200 filex**
3. $ **runit –xy**
4. $ **runit -*yx* -n 30**
5. $ **runit -n250 -xy *filmy***

(any other combination of these arguments)

설명

1. 프로그람 runit는 네개의 인수들을 가진다. x는 추가선택이고 n은 뒤에 수인수를 요구하는 추가선택이며 filex는 혼자 나타나는 인수이다.
2. 프로그람 runit는 추가선택 x, n 그리고 수인수 200을 조합한다.;
3. x와 y추가선택이 조합되여 프로그람 runit가 호출된다.
4. y와 x추가선택이 조합되여 프로그람 runit가 호출된다. n추가선택이 수인수 30과 함께 개별적으로 넘겨 진다.
5. 프로그람 runit는 수인수와 조합된 n추가선택, y추가선택과 조합된 x추가선택 그리고 분리된 filey로 호출된다.

runit프로그람에 대하여 상세히 설명하기전에 getopts가 사용되는 프로그람으로부터 그 행을 검사하여 인수들을 어떻게 처리하는가를 본다.

실례 8-103

(A Line from the Script Called “runit”)
while getopts :xyn: name

설명

- x, y 그리고 n은 추가선택들이다.
- 지령행에 입력된 추가선택들은 횡선으로 시작된다.
- 횡선을 포함하지 않는 임의의 추가선택들은 getopts에게 추가선택목록이 끝이라는것을 알려 준다.
- getopts는 호출될 때마다 자기가 찾는 다음번 추가선택을 횡선이 없이 변수 name에 넣는다(여기서 임의의 변수이름을 사용할수 있다.). 만일 비법

인수가 주어 지면 name에 물음표가 대입된다.

- OPTIND는 1로 초기화되고 getopts가 지령행인수처리를 끝낼 때마다 다음번에 처리할 인수의 번호로 증가되는 특수변수이다.
- OPTARG변수는 합법적인 인수의 값을 가진다.

getopst스크립트 다음의 실례들은 getopst가 인수들을 어떻게 처리하는가를 보여 준다.

실례 8-104

(The Script) $ **cat optsl**

```
#!/bin/sh
# Program opts1
# Using getopts -- First try --
1.  while getopts xy options
    do
2.      case $options in
3.        x) echo "you entered -x as an option";;
      y) echo "you entered -y as an option";;
      esac
    done
```

(The Command Line)

```
4.  $ opts1 -x
    you entered -x as an option
5.  $ opts1 -xy
    you entered -x as an option
    you entered -y as an option
6.  $ opts1 -y
    you entered -y as an option
7.  $ optsl -b
    opts1:   illegal option --  b
8.  $ opts1 b
```

설명

1. getopst지령은 while지령을 위한 조건으로 사용된다. 이 프로그람에 유효한 추가선택들이 getopst지령뒤에 현시된다. 그것들은 x와 y이다. 매 추가선택이 순환본체에서 하나하나 검사된다. 매 추가선택은 안내횡선이 없이 변수 options에 대입된다. 처리할 인수가 더이상 없을 때 getopst는 령이 아닌 상태로 탈퇴하여 while순환이 끝나게 한다.
2. case지령을 사용하여 options변수에 있는 가능한 추가선택 x 또는 y를 각각 검사한다.
3. 만일 x가 선택되였다면 문자렬 you entered x as an options이 현시된다.
4. 지령행에서 opts1스크립트에 getopts에 의해 처리되는 합법적인 추가선택인 x추가선택이 주어 진다.

5. 지령행에서 opts1스크립트에 xy추가선택이 주어 진다. x와 y는 getopt에 의해 처리되는 합법적인 추가선택이다.
6. 지령행에서 opt1스크립트에 getopts에 의해 처리되는 합법적인 추가선택인 y추가선택이 주어 진다.
7. opts1스크립트에 비법적인 추가선택인 b추가선택이 주어 진다. getopts는 표준오유에 오유통보를 보낸다.
8. 앞에 횡선이 붙지 않은 추가선택은 추가선택이 아니며 getopts가 인수처리를 그만두게 한다.

실례 8-105

(The script)
$ cat opts 2

```
#!/bin/sh
   # Program opts2
   # Using getopks -- Second try --
1.  while getopts xy options 2> /dev/null
    do
2.      case $options in
        x) echo "you entered -x as an option"; ;
        y) echo "you entered -y as an option"; ;
3.      \ ?) echo "Only -x and -y are valid options"   1>&2;;
        esac
    done
```

(The Command Line)

```
   $ opts2 -x
   you entered -x as an option

   $ opts2 -y
   you entered -y as an option

   $ opts2 xy

   $ opts2 -xy
   you entered -x as an option
   you entered -y as an option

4.  $ opts2 -g
    Only -x and -y are valid options

5.  $ opts2 -c
    Only -x and -y are valid options
```

설명

1. getopts로부터 오유통보가 있으면 그것은 /dev/null로 방향바꾸기된다.
2. 만일 추가선택이 적당한 추가선택이 아니면 물음표가 options변수에 대입된다. case지령을 물음표를 검사하는데 사용하여 사용자의 오유통보를 표준오

유에 출력할수 한다.

3. options변수에 물음표가 대입되면 case명령이 실행된다. 물음표는 거꿀빗선으로 보호되므로 쉘은 그것을 통용기호로 보지 않고 파일이름대입을 진행하려 한다.
4. g는 합법적인 추가선택이다. 물음표가 변수 options에 대입되고 오유통보가 현시된다.
5. c는 합법적인 추가선택이 아니다. 물음표가 변수 options에 대입되고 오유통보가 현시된다.

실례 8-106

(The Script)

```
$ cat opts3
    #!/bin/sh
    # Program opts3
    # Using getopts -- Third try --
1.   while getopts dq: options
     do
       case $options in
2.              d) echo "-d is a valid switch ';;
3.              q) echo "The argument for -q is $OPTARG";;
         \ ?) echo "Usage:opts3 -dq filename ..." 1>&2;;
       esac
    done
```

(The Command Line)

```
4.   $ opta3 -d
     #d is a valid switch

5.   $ opts3 -g   foo
     The argument for -q is foo

6.   $ opts3 -q
     Usage:opts3 -dq filename ...

7.   $ opts3 -e
     Usage:opts3 -dq filenaue ...

8.   $ opts3 e
```

설명

1. while지령이 getopts의 탈퇴상태를 검사한다. 만일 getopts가 인수를 성과적으로 처리할수 있으면 령탈퇴상태를 되돌리고 while순환이 시작된다. 인수목록에 붙은 옹근두점은 g추가선택이 인수를 요구한다는것을 나타낸다. 인수는 특수변수 OPTARG에 기억된다.
2. d는 합법적인 추가선택들중의 하나이다. 만일 d가 추가선택으로 입력되면 d(횡선이 없이)는 options변수에 기억된다.

3. q는 합법적인 추가선택들중의 하나이다. q추가선택은 인수를 요구한다. q추가선택과 인수사이에 공백이 있어야 한다. 인수뒤에 오는 q가 추가선택으로 입력되면 q는 안내횡선이 없이 변수 options에 기억되고 인수는 OPTARG 변수에 기억된다. q추가선택뒤에 인수가 없으면 물음표가 변수 options에 기억된다.
4. d추가선택은 opts3에 대한 합법적인 인수이다.
5. 인수를 가지는 q추가선택도 opts3에 대한 합법적인 인수이다.
6. q추가선택은 인수가 없을 때 오류로 된다.
7. e추가선택은 무효이다. 물음표는 추가선택이 비합법적일 때 변수 options에 기억된다.

실례 8-107

```
$ cat opts4
    #!/bin/sh
    # Program opts4
    # Using getopts – Fourth try –
1.  while getopts xyz: arguments 2>/dev/null
    do
        case $arguments   in
2.      x)  echo "you entered –x as an option. ";;
        y)  echo "you entered –y as an option. ";;
3.      z)  echo "you enetred –z as an opyion. ";;
              echo "\ $OPTARG is $OPTARG.";;
4.      \ ?) echo "Usage opts4   [-xy]   [-z    argument]"
              exit  1;;
        esac
        done
5.  echo  "The initial value of \ $OPTIND is 1.
        The final value of \ $OPTIND is $OPTIND.
        Scince this reflects the number of the next command line
        argument,the number of arguments passed was
        `expr $OPTIND – 1`. "
```

(The Command Line)

```
$ opts4 –xyz foo
you entered –x as an opyion.
you entered –y as an option.
you entered –z as an option.
$OPTARG is foo.
The initial value of $OPTIND is 1.
The final value of $OPTIND is 3.
Since this refelects the number of the next command
Line argument, the number of arguments passed was 2.
$ opts4 –x –y –z   boo
```

you entered –x as an option.
you entered –y as an option.
you entered –z as an option.
The initial value of $OPTIND is 1.
The final value of $OPTIND is 5.
Since this refelects the number of the next command
Line argument, the number of arguments passed was 4.
$ **opts4 –d**
Usage: opts4 [-xy] [-z argument]

설명

1. while지령은 getopts의 탈퇴상태를 검사한다. getopts가 인수를 성과적으로 처리할수 있으면 령탈퇴상태를 되돌리고 while순환이 시작된다. z추가선택에 붙은 옹근두점은 getopts에게 인수가 –z추가선택다음에 놓여야 한다고 알려 준다. 추가선택이 인수를 가지면 그 인수는 getopts의 내부지령변수 OPTARG에 기억된다.
2. x가 추가선택로 주어 지면 변수 argments에 기억된다.
3. 인수가 있는 z가 추가선택로 주어 지면 그 인수는 내부지령변수 OPTARG에 기억된다.
4. 무효인 추가선택이 입력되면 물음표가 변수 argments에 기억되고 오유통보가 현시된다.
5. 특수 getopts변수인 OPTIND에는 다음번에 처리될 추가선택의 번호가 들어 있다. 그 값은 지령행인수들의 실지수보다 항상 한개 더 많다.

12. eval지령과 지령행구문해석

eval지령을 사용하여 지령행을 평가하고 모든 쉘대입들을 진행하며 지령행을 실행한다. 지령행의 표준구문해석이 불충분할 때 사용한다.

실례 8-108

1. $ **set a b c d**
2. $ **echo The last argument is 全$$#**
3. *The last argument is $4*
4. $ **eval echo The last argument is $$#**
   The last argument is d
5. $ **set -x**
   $ **eval echo The last argument is $$#**
   +eval echo the last argument is $4
   +echo the last argument is d
   The last argument is d

설명

1. 4개의 위치파라메터들이 설정된다.

2. 요구되는 결과는 마지막위치파라메터의 값을 인쇄하는것이다. $는 화페기호를 문자그대로 인식한다. $#는 위치파라메터의 개수 4와 같다. 쉘은 $#를 평가한후 $4의 값을 얻기 위하여 행을 다시 구문해석하지 않는다.
3. $4가 마지막인수대신 인쇄된다.
4. 쉘이 변수바꿔넣기를 진행한후 eval지령이 변수바꿔넣기를 진행하고 그다음에 echo지령을 실행한다.
5. echoing을 설정하여 구문해석하는 순서를 보여 준다.

실례 8-109

```
(From SVR4 Shutdown Program)
1.  eval '/usr/bin/id | /usr/bin/sed 's/[^a-z0-9=].*//''
2.  if [ "${uid:=0}" -ne 0 ]
    then
3.      echo $0: Only root can run $0
     exit 2
    fi
```

설명

1. id프로그람의 출력이 sed에 보내져 문자렬에서 uid부분을 뽑아 낸다. id에 대한 출력은 다음과 같다.

   uid=9496(ellie)gid=40 groups=40
   uid=0(뿌리) gid=1(daemon) groups=1(daemon)

   sed의 정규식은 다음과 같다. 한개 자모문자, 수 또는 같기부호가 아닌 임의의 문자를 찾아서 그 문자와 뒤에 오는 모든 문자들을 지운다. 즉 첫번째 열린 괄호로부터 행의 끝까지 모든것을 지운다. 그 결과는 uid=9496 또는 uid=0이다. eval은 지령행을 평가한후 다음의 지령을 실행한다.

   uid=9496 또는 uid=0

   실례로 사용자의 ID가 뿌리이면 실행되는 지령은 uid=0이다. 쉘에 uid라고 하는 국부변수를 작성하고 령을 대입한다.
2. 지령변경자를 사용하여 uid변수의 값이 령인가를 검사한다.
3. uid가 령이 아니면 echo지령이 스크립트이름($0)과 통보를 현시한다.

13. 쉘호출추가선택

쉘은 sh지령에 의해 기동될 때 추가선택을 사용하여 자기의 동작을 변경시킬수 있다(표 8-17).

표 8-17. 쉘호출추가선택

추가선택	의 미
-i	쉘이 대화형방식에 있다. QUIT와 INTERRUPT가 무시된다.
-s	지령들이 표준입력으로부터 읽어 지고 출력은 표준오유로 보내진다.
-c string	지령들은 문자렬로부터 읽어 진다.

14. set지령과 추가선택

set지령을 사용하여 쉘추가선택들을 지령행인수들을 처리할 때와 같이 절환할수 있다. 추가선택을 설정하려면 횡선(-)이 추가선택앞에 붙어야 하고 설정하지 않으려면 더하기부호(+)를 추가선택앞에 붙여야 한다. set추가선택들의 목록을 표 8-18에서 참고하시오.

실례 8-110

```
1.  $ set -f
2.  $ echo *
     *
3.  $ echo ??
     ??
4.  $ set +f
```

설명

1. f추가선택이 설정되여 파일이름이 확장되지 않게 한다.
2. 별표가 확장되지 않는다.
3. 물음표가 확장되지 않는다.
4. f가 해제된다. 파일이름을 확장할수 있다.

표 8-18. set지령추가선택

추가선택	의 미
-a	수정되거나 넘겨 진 변수들을 나타낸다.
-e	지령이 령이 아닌 상태를 복귀하면 프로그람에서 벗어 난다.
-f	파일이름확장을 할수 없게 한다.
-h	함수지령들을 실행될 때가 아니고 정의 될 때 함수들로 취급한다.
-k	모든 예약어인수들은 지령이름앞에 놓이는것들이 아니라 지령에 대한 환경에 놓인다.
-n	지령을 읽지만 그것을 실행하지 않는다. 오유수정에 사용된다.
-t	한개 지령을 읽고 실행한후에 탈퇴한다.
-u	대입을 진행할 때 unset변수를 오유로 취급한다.
-v	쉘입력행들을 입력된대로 인쇄한다. 오유수정에 사용된다.
-x	지령이 실행될 때 지령들과 그 인수들을 인쇄한다. 오유수정에 사용된다.
- -	임의의 기발들도 변화시키지 않는다.

15. 쉘의 내부지령

쉘에는 그의 원천코드에 내장된 많은 지령들이 있다. 지령들이 내장되여 있기때문에 쉘은 그것들을 디스크에서 찾지 않고 실행속도를 더 높일수 있다.

표 8-19에서 내부지령들을 보여 주었다.

표 8-19. 내부지령

지 령	기 능
:	아무것도 하지 않는 지령이다. 탈퇴상태를 령으로 넘긴다.
.file	점지령은 파일로부터 지령을 읽어 실행시킨다.
break[n]	순환을 보시오.
continue[n]	순환을 보시오.
cd	등록부를 변화시킨다.
echo[args]	인수들을 출력한다.
eval command	쉘은 실행전에 지령행을 두번 조사한다.
exec지령	쉘에서 지령을 실행한다.
exit[n]	상태를 n으로 하여 쉘을 탈퇴한다.
export[var]	var가 보조쉘들에 알려 지게 한다.
hash	지령들을 더 빨리 탐색하기 위하여 내부하쉬표를 조종한다.
kill [-signal process]	신호를 프로쎄스의 PID번호나 과제번호에 보낸다. 신호들의 목록에 대하여 usr/include /sys/signal.h를 보시오.
Getopts	쉘스크립트들에서 사용되여 지령행을 구문해석하고 합법적인 스위치들을 검사한다.
login[username]	체계에 등록한다.
newgrp[arg]	실지그룹과 효률적인 그룹 ID를 변화시켜 사용자를 새로운 그룹에 등록한다.
pwd	현재작업등록부를 인쇄한다.
read[var]	표준입력으로부터 변수 var에로 행을 읽어 들인다.
readonly[var]	변수 var를 읽기전용함수로 만든다. 재설정될수 없다.
return[n]	n이 되돌이값으로 주어 진 탈퇴값으로 되는 함수로부터 복귀한다.
set	표 8-18을 보시오.
shift[n]	위치파라메터들을 n번 왼쪽으로 밀기한다.
stop pid	프로쎄스번호 PID의 실행을 정지시킨다.
suspend	현재쉘의 실행을 중지시킨다(등록가입쉘인 때는 중지시키지 않는다.).
times	이 쉘로부터 실행되는 프로쎄스들에 대한 축적된 사용자와 체계시간을 인쇄한다.
trap[arg][n]	쉘이 신호n(0, 1, 2 또는 15)을 받을 때 arg를 실행시킨다.

(표계속)

지 령	기 능
type[command]	지령의 형태를 인쇄한다. 실례: ksh에서 pwd는 지령 whence-v에 대한 별명을 가진다.
umask[octaldigits]	소유자, 그룹 등에 대한 사용자파일작성방식마스크
unset[name]	변수나 함수의 해제된 값
wait[pid#n]	PID번호 n을 가진 배경프로쎄스를 대기하며 끝상태를 알린다.
ulimit[options size]	프로쎄스들에 대한 최대한계를 설정한다.
umask[mash]	인수없이 허가를 위한 파일작성마스크를 인쇄한다.

BOURNE쉘련습

련습문제 8: 기동

1. 어떤 프로쎄스가 현시장치에 등록가입재촉문을 내보내는가?
2. 어떤 프로쎄스가 HOME, LOGNAME, PATH에 값을 대입하는가?
3. 사용자가 어느 쉘을 사용하고 있는지 어떻게 아는가?
4. 등록가입쉘이 어디에(무슨 파일에) 대입되는가?
5. /ete/profile과 .profile사이의 차이를 설명하시오. 어느것이 먼저 실행되는가?
6. .profile파일을 다음과 같이 편집하시오.

   ㄱ. 사용자를 맞이하시오.
   ㄴ. 홈등록부가 경로에 없으면 거기에 추가하시오.
   ㄷ. stty를 사용하여 Backspace건에 지우기기능을 설정하시오.
   ㄹ. profile을 입력하시오. 점지령의 기능은 무엇인가?

련습문제 9: 쉘메타문자

1. wildcards라고 하는 등록부를 만드시오. 그 등록부로 등록부를 변화시키고 재촉문에서 다음의것을 입력하시오.
   touch ab abc a1 a2 a3 a11 a12 ba ba.1 ba.2 filex filey abc ABC ABC2 abc
2. 다음의것을 수행하는 지령을 작성하고 검사하시오.

   ㄱ. a로 시작하는 모든 파일들을 현시하시오.
   ㄴ. 적어도 한개의 아라비아수자로 끝나는 모든 파일을 현시하시오.
   ㄷ. a나 A로 시작되는 모든 파일을 현시하시오.
   ㄹ. 마지막에 한개의 아라비아수자와 그앞에 점이 오는 모든 파일들을 현시하시오.

ㅁ. 2개의 a문자를 가지는 모든 파일들을 현시하시오.
ㅂ. 모든 문자가 대문자이고 3개문자로 된 파일들을 현시하시오.
ㅅ. 10, 11, 12로 끝나는 파일들을 현시하시오.
ㅇ. x나 y로 끝나는 파일들을 현시하시오.
ㅈ. 한개의 아라비아수자, 대문자 또는 소문자로 끝나는 모든 파일들을 현시하시오.
ㅊ. ab 또는 B로 시작하지 않는 모든 파일들을 현시하시오.
ㅋ. a나 A로 시작하면서 2개 문자로 된 파일을 지우시오.

련습문제 10: 방향바꾸기

1. 말단과 련관된 3개의 파일흐름들의 이름이 무엇인가?
2. 파일서술자란 무엇인가?
3. 다음의 문제를 수행하기 위해 어떤 지령을 사용하여야 하는가?

ㄱ. ls지령의 출력을 lsfile이라고 하는 파일에 방향바꾸기하기 위해서는 어떤 지령을 사용하여야 하는가?
ㄴ. date지령의 출력을 lsfile에 방향바꾸기하고 추가하기 위해서는 어떤 지령을 사용하여야 하는가?
ㄷ. who지령의 출력을 lsfile에 방향바꾸기하기 위해서는 어떤 지령을 사용하여야 하는가?

4. 다음의것을 하시오.

ㄱ. cp만을 입력하시오. 무슨 일이 생기는가?
ㄴ. 우의 실례로부터 생기는 오유를 파일에 기억하시오.
ㄷ. find지령을 사용하여 현재어미등록부에서 파생되고 등록부형인 모든 파일들을 찾으시오. 표준출력을 find파일에 보존하고 오유는 find.errs파일에 보존하시오.
ㄹ. 3개 지령들의 출력을 gottem_all이라고 부르는 파일로 방향바꾸기하시오.
ㅁ. ps와 wc지령과 함께 파이프(들)를 사용하여 현재 몇개 프로쎄스를 실행하고 있는가를 알아 보시오.

련습문제 11: 첫번째 스크립트

1. 다음의것을 하는 greetme이라고 부르는 스크립트를 작성하시오.

ㄱ. 사용자의 이름, 스크립트의 이름, 목적을 나타내는 설명문부분을 포함하시오.
ㄴ. 사용자를 맞이하시오.
ㄷ. 날자와 시간을 인쇄하시오.
ㄹ. 이 달의 력서를 인쇄하시오.
ㅁ. 콤퓨터이름을 인쇄하시오.
ㅂ. 이 조작체계의 이름과 계렬을 인쇄하시오(cat/etc/motd).

ㅅ. 어미등록부에 있는 모든 파일들의 목록을 인쇄하시오.
ㅇ. 뿌리가 실행하고 있는 모든 프로쎄스들을 인쇄하시오.
ㅈ. TERM, PATH, HOME변수들의 값을 인쇄하시오.
ㅊ. 디스크사용법(du)을 인쇄하시오.
ㅋ. id지령을 사용하여 그룹 ID를 인쇄하시오.
ㅌ. Korea is a land of morning calm를 인쇄하시오.
ㅍ. 사용자에게 인사를 하고 현재시간을 알려 주시오(date지령에 대해 사용지도서페지를 보시오.).

2. 스크립트가 실행형이 되게 하시오.

   chmod +x greetme

3. 스크립트의 첫번째 행은 무엇이였는가? 왜 이 행이 필요한가?

련습문제 12: 지령행인수

1. 2개의 인수들을 가지는 rename이라고 부르는 스크립트를 작성하시오. 첫번째 인수는 초기파일의 이름이고 두번째 인수는 파일의 새 이름이다. 만일 사용자가 2개의 인수들을 제공하지 않으면 사용통보가 현시장치에 나타나고 스크립트가 탈퇴한다. 다음에 스크립트가 어떻게 실행되는가를 보여 준다.

   $ **rename**
   Usage: rename oldfilename newfilename
   $

   $ **rename file1 file2**
   file1 has been renamed file2
   Here is a listing of the directory:
   a file2
   b file.bak

2. 다음의 find지령(SunOS)은 100K보다 더 크고 지난 주에 수정된 뿌리부분에 있는 모든 파일들을 현시한다(체계에서 정확한 find문법을 위해 안내페지를 검사하시오.).

   find / -xdev -mtime -7 -size +200 -print

   2개의 인수를 가지는 bigfiles라고 하는 스크립트를 작성하시오. 하나는 mtime, 다른 하나는 size값이다. 사용자가 2개의 인수를 제공하지 않으면 적당한 오유통보문이 표준오유에 보내진다.

3. 시간이 있으면 vi에 대한 여벌파일을 작성하는 vib라고 하는 스크립트를 작성하시오.
   여벌파일은 초기이름에 확장자 .bak가 붙은것이다.

련습문제 13: 사용자입력받기

1. 다음의것을 하는 nosy라고 하는 스크립트를 작성하시오.

ㄱ. 사용자의 완전이름 즉 성과 이름을 물어 보시오.
ㄴ. 사용자의 첫번째 이름으로 그를 맞이하시오.
ㄷ. 사용자의 출생년도를 묻고 나이를 계산하시오(expr를 사용하시오.).
ㄹ. 사용자의 등록가입이름을 묻고 사용자 ID를 인쇄하시오(/etc/passwd 로부터).
ㅁ. 사용자에게 그의 홈등록부를 알려 주시오.
ㅂ. 사용자에게 그가 실행하고 있는 프로쎄스를 보여 주시오.
ㅅ. 사용자에게 파일과 현재시간을 알려 주시오. 출력은 아래와 같은 형식으로 되여야 한다.

+day of the week is Tuseday and the current time is 04:07:38 PM.

2. datafile이라고 하는 본문파일을 작성하시오(파일이 이미 존재하지 않는다면). 매 입력은 옹근두점에 의해 분리되는 마당들로 구성된다. 마당들은 다음과 같다.

ㄱ. 성과 이름
ㄴ. 전화번호
ㄷ. 주소
ㄹ. 생년월일
ㅁ. 로임

3. 다음의것을 하는 lookup이라고 하는 스크립트를 작성하시오.

ㄱ. 스크립트이름, 사용자이름, 날자 그리고 이 스크립트를 작성하는 리유를 밝힌 설명문을 포함하시오. 이 스크립트를 작성하는 목적은 datafile을 분류된 순서로 현시하는것이다.
ㄴ. datafile을 마지막이름으로 분류하시오.
ㄷ. 사용자에게 datafile의 내용을 보여 주시오.
ㄹ. 사용자에게 파일에 있는 입력들의 개수를 알려 주시오.

4. -x와 -v추가선택들을 설정하여 스크립트를 오유수정하시오. 이 지령들을 어떻게 사용했는가? 그것들은 어떻게 다른가?

련습문제 14: 조건명령

1. 다음의것을 수행하는 checking이라고 하는 스크립트를 작성하시오.

ㄱ. 지령행인수를 취하시오. 그것은 사용자의 등록가입이름이다.
ㄴ. 지령행인수가 주어 졌는가를 검사해 보시오.
ㄷ. 사용자가 /etc/passwd파일에 있는가를 검사해 보시오. 만일 그렇다면 다음의것을 인쇄하시오.
found<user>in the /etc/passwdfile.
그렇지 않으면 아래와 같이 인쇄한다.
no such user on our system.

2. lookup스크립트에서 사용자에게 datafile에 입력을 추가하려는가를 물어 보시오. 만일 그 대답이 yes나 y이면

ㄱ. 사용자가 새로운 이름, 전화번호, 주소, 생년월일, 로임을 입력하도록 재촉하시오. 매 항목이 적당한 변수에 기억된다. 마당들사이에 옹근두점을 넣어 정보를 datafile에 추가한다.

ㄴ. 파일을 마지막이름들로 분류하시오. 사용자에게 입력을 추가했다는것을 알려 주고 행번호가 앞에 붙은 행을 보여 주시오.

련습문제 15: 조건부와 파일시험

1. cheking을 다시 작성하시오. 이름 지어 진 사용자가 /etc/passwd파일에 있는가를 검사한후 프로그람은 사용자가 등록가입되였는가를 검사해 본다. 만일 그렇다면 프로그람은 실행되는 모든 프로쎄스들을 인쇄한다. 그렇지 않으면 사용자에게 다음과 같이 알린다.

   <user> is not logged on.

2. lookup스크립트의 실행은 datafile에 의존한다. lookup스크립트에서 datafile이 존재하는가, 그것을 읽을수 있는것인가, 쓸수 있는것인가를 검사해 보시오.

   lookup스크립트에 다음과 같은 차림표를 추가하시오.

   [1]입력추가
   [2]입력소거
   [3]입력보기
   [4]탈퇴

   ㄱ. 이미 작성된 스크립트에 입력추가차림표가 있다. 입력추가프로그람은 그 이름이 datafile에 이미 있는가를 검사하고 있으면 그것을 사용자에게 알려 주는 코드를 포함해야 한다. 만일 이름이 없다면 새로운 입력을 추가한다.

   ㄴ. 입력지우기, 입력보기, 탈퇴기능을 위한 코드를 작성한다.

   ㄷ. 스크립트의 지우기부분은 먼저 입력을 지우기전에 그것이 존재하는가를 검사해야 한다. 존재하지 않으면 사용자에게 통지한다. 그렇지 않으면 입력을 지우고 사용자에게 그것을 알려 준다. 탈퇴할 때 아라비아수자로 적합한 탈퇴상태를 정확히 표시하도록 하시오.

   ㄹ. 지령행으로부터 탈퇴상태를 어떻게 검사하는가?

련습문제 16: case명령

1. ps지령은 UCB와 AT&T UNIX에서 다르다. AT&T UNIX에서 모든 프로쎄스들을 현시하는 지령은 다음과 같다.

   ps-ef

   UCB UNIX에서는

   ps-aux

   여러가지 다른 체계형들을 검사하는 systype라는 프로그람을 작성하시오. 검사될 조항은 다음과 같다.

AIX
IRIX
HP-UX
SCU
OSF1
ULTRIX
SunOS(Solaris/SunOS)
OS

Solaris, HP-UX, SCO와 IRIx는 AT&T형체계들이다. 나머지는 BSDish이다. 지금 사용하고 있는 UNIX의 판본이 표준출력으로 인쇄된다. 체계이름은 uname-s지령으로 찾거나 또는 /etc/motd파일로부터 찾아 볼수 있다.

련습문제 17: 순환

다음의것들중 하나를 선택하시오.

1. 새로운 우편을 검사하고 그것이 도착했으면 현시장치에 통보를 현시하는 mchecker라는 프로그람을 작성하시오.
   ㄱ. 프로그람은 사용자에 대한 우편스풀파일의 크기를 얻는다(스플파일은 AT&T체계에서는 /usr/mail/$LOGNAME에 있고 UCB체계에서는 /usr/spool/mail /$USER에 있다. 만일 그 파일을 찾을수 없으면 find 지령을 사용하시오.). 스크립트는 30초에 한번씩 련속순환에서 실행한다. 순환이 진행될 때마다 우편스플파일의 크기를 이전순환때의 크기와 비교한다. 만일 새 크기가 낡은 크기보다 크면 현시장치에 통보 Username, You have now mail을 인쇄하시오. 파일의 크기는 find 지령으로부터 또는 ls-l, wc-c로부터 나오는 출력을 보고 알수 있다.
2. 사용자들의 목록에 한번에 한개씩 그들이 현재 사용하고 있는 블로크수의 목록을 전송하는 dusage라는 프로그람을 작성하시오. 사용자들의 목록은 potential_hogs라는 파일에 있다.
   potential_hogs파일에 있는 사용자들중 하나가 admin이다.
   ㄱ. potential_hogs파일이 존재하며 읽을수 있는가를 파일시험을 사용하여 검사하시오.
   ㄴ. 순환을 사용하여 사용자들의 목록을 반복한다. 500개이상의 블로크를 사용하고 있는 사용자들에게만 우편을 전송한다. 사용자 admin을 건너 뛴다(즉 그는 우편통보를 받지 못한다.).
   우편통보는 dusage스크립트의 here문서에 기억된다.
   ㄷ. 우편을 받은 매 사람들의 이름목록을 보존하시오. log파일을 작성하여 이것을 하시오. 목록에 있는 매 사람에게 우편이 전송된 다음 우편을 받은 사람들의 수와 그 이름목록을 인쇄하시오.

련습문제 18: 함수

1. Lab9에서 작성한 systype프로그람을 체계의 이름을 되돌리는 함수로 다시 작성하시오. 이 함수를 사용하여 checking프로그람에서 ps지령을 어느 추가선택과 함께 사용하겠는가를 결정하시오.

2. AT&T UNIX에서 모든 프로쎄스들을 현시하는 ps지령은 다음과 같다.

   ps-ef

3. BSD UNIX에서 지령은

   ps-aux

   이다.

4. 모든 림시파일들을 지우고 스크립트를 벗어 나는 cleanup이라는 함수를 작성하시오. 만일 프로그람이 실행중일 때 새치기나 정지신호가 보내지면 trap지령이 cleanup함수를 호출한다.

5. here문서를 사용하여 lookup스크립트에 다음과 같은 새로운 차림표항목을 추가하시오.

   [1] 입력추가
   [2] 입력지우기
   [3] 입력변경
   [4] 입력보기
   [5] 탈퇴

   차림표에 있는 매 항목들을 처리하는 함수를 작성하시오. 사용자가 유효입력을 선택하고 함수가 실행된 다음 사용자가 차림표를 다시 보려고 하는가를 질문하시오. 만일 무효입력이 입력되면 프로그람은 다음과 같이 인쇄된다.

   Invalid entry, try again

   그다음 차림표가 다시 현시된다.

6. lookup스크립트에서 입력보기밑에 부분차림표를 작성하시오.
   사용자에게 선택된 인물에 대한 정보를 보려고 하는가를 물어 본다.

   ㄱ) 전화번호
   ㄴ) 주소
   ㄷ) 생년월일
   ㄹ) 로임

7. trap지령을 사용하여 프로그람실행중에 새치기신호가 전송되면 스크립트에서 지우기동작을 진행하시오.

제 9 장. C쉘

제 1 절. 대화형C쉘

C쉘이 재촉문을 현시하기전에 몇개의 프로쎄스들이 먼저 실행된다(그림 9-1).

1. 기동

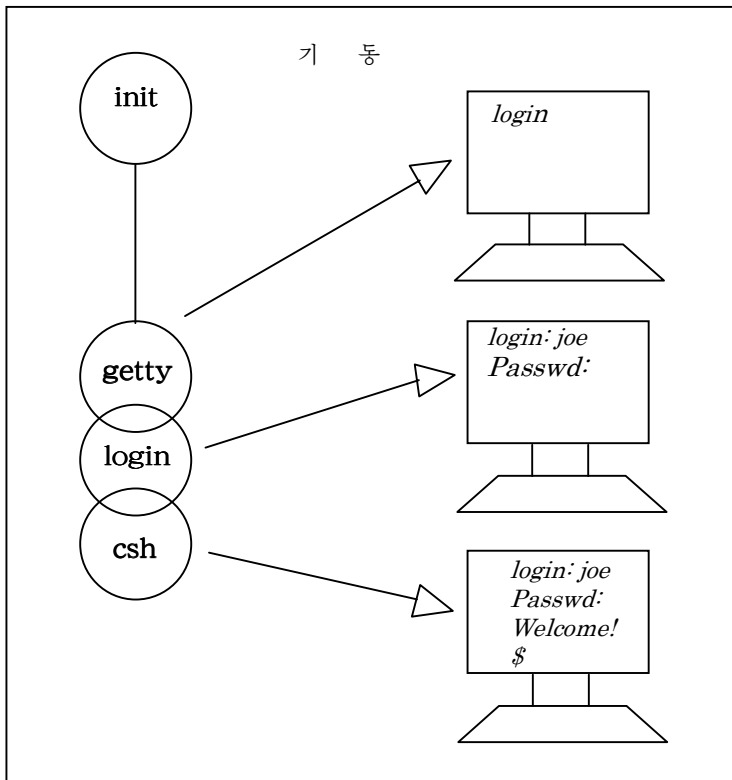



그림 9-1. 체계기동과 C 쉘

체계가 기동한후 처음으로 실행되는 프로쎄스를 init라고 한다. 여기에 프로쎄스식별번호(PID)1이 대입된다. 그것은 inittab(체계 V)라고 하는 파일로부터 명령을 받는다. 즉 getty프로쎄스(BSD)를 파생시킨다. 이 프로쎄스들은 말단포구들을 열어 입력이 표준입력(stdin)으로부터 들어 오고 출력이 표준출력(stdout)과 표준오유 (stderr)로 나갈수 있게 하며 login재촉문을 현시장치에 현시한다. 그다음에 /bin/login프로그람이 실행된다. 등록가입프로그람은 암호를 재촉하고 암호를 입력하면 그것을 확인한 다음에 초기작업환경을 설정하고 쉘 즉 /bin/csh를 초기화한다. C쉘은 사용자의 홈등록부에서 .cshrc라고 하는 파일을 찾는데 이것은 현재 작업하고 있는 C쉘환경을 임의의 요구대로 수정할수 있게 하는 초기화파일이다.

.cshrc파일에 있는 지령들이 실행된후 .login파일에 있는 지령들이 실행된다.

.cshrc파일은 새로운 C쉘이 기동될 때마다 실행된다. .login파일은 사용자가 등록가입할 때에만 한번 실행되고 사용자의 환경을 초기화하는 지령들과 변수들을 가지고 있다. 이 파일들로부터 지령들이 실행된후 재촉문 %가 현시장치에 나타나고 C쉘이 지령을 대기한다.

2. 환경

초기화파일 csh프로그람이 시작된후에 사용자의 홈등록부에 있는 2개 파일들 즉 .cshrc파일과 .login파일이 실행되게 된다. 이 파일들은 사용자가 자기의 환경을 초기화할수 있게 한다.

.cshrc파일 .cshrc파일은 C쉘변수설정을 포함하고 csh보조쉘이 파생될 때마다 실행된다. 별명들과 리력이 보통 여기서 설정된다.

실례 9-1

```
(The .cshrc File)
1.  if ( $?proinpt ) then
2.      set prompt = “\ ! stardust > ”
3.      set history = 32
4.      set savehist =5
5.      set noclobb
6.      set filec fignore = ( .o )
7.      set cdpath = ( /home/jody/ellie/bin /usr/local/bin /usr/bin )
8.      set ignoreeof
9.      alias m more
        alias status ‘date;du –s’
        alias cd ‘cd \ !*;set prosipt = “\ ! <$cwd> ”’
    endif
```

설명

1. 재촉문이 ($?prompt)로 설정되여 있으면 쉘은 대화형으로 동작한다. 즉 그것은 스크립트안에서 실행하지 않는다.
2. 1차재촉문은 현재리력사건의 번호, 이름인 stardust 그리고 >문자의 조합으로 설정된다. 이것은 기정값인 %재촉문을 변화시킨다.
3. history변수에 32가 설정된다. 이것은 현시장치에 나타나는 리력사건들의 수를 조종한다. 제일 최근에 입력했던 32개 지령들이 history를 입력할 때 현시된다.
4. 등록탈퇴할 때 보통 리력목록이 지워 진다. savehist변수를 사용하여 리력목록끝으로부터 지적된 수의 지령들을 기억할수 있다. 이 실례에서 홈등록부에 있는 history파일에 제일 최근의 5개 지령들을 기억하여 다시 등록가입할 때 쉘은 그 파일이 존재하는가를 검사하고 기억된 리력행들을 새로운 리력목록의 제일 우에 놓는다.
5. noclobber변수를 설정하여 사용자가 방향바꾸기를 사용할 때 부주의로 파

일들을 지우지 않게 할수 있다. 실례로 sort myfile>myfile은 myfile을 지운다. 그러나 noclobber를 설정하면 통보 file exists가 현시장치에 나타난다.

6. 파일이름완성을 위해 filec변수를 사용하여 파일이름에서 제일 중요한 첫번째 문자들만을 입력하고 ESC건을 누르면 쉘이 파일이름의 나머지를 완성한다. 파일이름을 입력할 때 ^D(Control-D)건을 누르면 C쉘은 그 문자렬과 일치하는 파일들의 목록을 현시한다. fignore변수를 사용하여 요구하는 파일이름들이 완성되지 않게 한다. 이 경우 모든 .o파일이름들(대상파일들)에 filec가 설정되였지만 그로부터 영향을 받지 않는다(328폐지에서《파일이름 완성: filec변수》를 참고하기 바란다.).
7. cdpath변수에 경로요소들의 목록이 대입된다. 등록부들을 변화시킬 때 등록부의 이름이 지정되고 그 등록부가 현재작업등록부의 부분등록부가 아니면 쉘은 cdpath등록부입력들을 조사하여 그속에서 지정된 등록부를 찾으면 그것을 변화시킨다.
8. ignoreeof변수는 ^D건을 눌러서 등록탈퇴하지 못하게 한다. mail프로그람과 같이 건반으로 입력을 받는 UNIX편의프로그람들은 ^D를 눌러서 끝낸다. 속도가 느린 체계에서 사용자는 흔히 ^D를 한번이상 누르게 된다. 첫번째 눌렀을 때 mail프로그람이 끝나고 두번째 눌렀을 때 사용자가 등록탈퇴한다. 그러나 ignoreeof를 설정하면 logout를 입력하여 탈퇴할수 있다.
9. 별명들을 설정하여 한개 지령이나 지령들의 묶음을 속기법으로 표현할수 있다. 별명을 입력하면 그에 대응한 지령들이 실행된다. more지령에 대한 별명은 m이다. m을 입력할 때마다 more지령이 실행된다. status별명은 날자와 사용자디스크의 총 사용범위를 인쇄한다. 별명 cd는 사용자가 등록부를 변화시킬 때마다 새로운 재촉문을 작성한다. 새로운 재촉문은 현재리력사건의 번호 (\ !*)와 < >로 둘러 싸이는 현재작업등록부 $cwd를 포함한다(319폐지에서 《별명》을 보기 바란다.).

.login파일 .login파일은 처음에 등록가입할 때 한번 실행된다. 보통 여기서 환경변수들과 말단을 설정한다. 이것은 windows응용프로그람들이 보통 시작되는 파일이다. 환경변수들은 쉘에서 파생된 프로세스들에 의해 계승되므로 한번만 설정하면 되고 말단설정들은 매 프로세스에 대해 재설정되지 않고 다 .login파일에서 설정된다.

실례 9-2

```
(The .login File)
1.  stty -istrip
2.  stty erase ^H
3.  #
    # If possible start the vindous system.
    # Give a user a chance to bail out
    #
4.  if ( 'tty' = = "/dev/console" ) then
5.     if ( $TERM == "sun- | | STERM == "AT386" ) then
6.           if ( ${?OPENWINHOME} == 0) then
```

```
 7.                 seter.-.        .SQ/usr/openwin
 8.             endif
          echo “”
 9.       echo -n “Starting OpenWindows in 5 seconds\
             (type Control-C to interrupt)”
10.       sleep 5
          echo “”
11.       $OPENWINHOME/bin/operwin
12.       clear
13.       logout
       endif
```

설명

1. stty지령이 말단에 대한 추가선택을 설정한다. 만일 -istrip가 추가선택이면 입력문자들은 7개의 비트들로 끊어 지지 않는다.
2. stty지령은 문자들을 지우는 Backspace건(Control-H)을 설정한다.
3. #로 시작되는 행들은 설명문이다. 그것은 실행명령이 아니다.
4. 현재말단창문 (tty)이 조종탁(/dev/console)이면 다음행이 실행된다. 그렇지 않으면 프로그람조종이 마지막 endif로 넘어 간다.
5. TERM변수의 값이 sun이나 AT386이면 다음행이 실행된다.
6. OPENWINHOME환경변수가 설정되지 않았으면 (설정되지 않았으면 $?는 0, 설정되였으면 1)다음행이 실행된다.
7. OPENWINHOME 환경변수가 /usr/openwin으로 설정된다.
8. 이 endif가 5행에 있는 if를 끝낸다.
9. 이 행이 현시장치에 현시되여 5초내에 Control-C를 누르지 않으면 Open Windows가 시작된다는것을 사용자에게 알려 준다.
10. 프로그람이 5s동안 잠시 중지된다.
11. openwin프로그람이 기동된다. 말단창문들의 모임(쉘과 지령도구창문들 그리고 조종탁창문)이 현시장치에 나타난다.
12. 창문을 닫으면 현시장치가 지워 진다.
13. 사용자가 등록탈퇴된다.

endif탐색경로 쉘은 path변수를 사용하여 지령행에 입력된 지령들을 찾는다. 탐색은 왼쪽에서 오른쪽으로 한다. 점은 현재작업등록부를 현시한다. 경로에 현시된 등록부들이나 현재작업등록부에 지령이 없으면 쉘은 통보 Command not found를 표준오유로 보낸다. 경로는 보통 .login파일에서 설정된다.[1] C쉘에서 탐색경로설정은 Bourne와 Korn쉘에서 다르다.

매 요소들은 공백에 의해 분리된다.

```
set path=(/usr/bin/usr/ucb/bin/usr.)
echo $path
/usr/bin/usr/ucb/bin/usr.
```

[1] path 변수를 .cshrc 파일에서 설정되는 cdpat 변수와 혼돈하지 말아야 한다.

환경변수 PATH는 다음과 같이 현시된다.

echo $PATH
/usr/bin:/usr/ucb:/bin/:.

C쉘은 내적으로 PATH가 이 쉘로부터 시작되고 path변수를 사용할 필요가 있는 다른 프로그람 (Bourne 또는 Korn쉘들과 같은)과 호환성을 유지하도록 환경변수를 갱신한다.

rehash지령 쉘은 탐색경로에 현시된 등록부들의 내용들로 이루어 지는 내부하쉬표를 만든다(만일 점이 탐색경로에 있다면 현재등록부인 점등록부에 있는 파일들은 하쉬표에 놓이지 않는다.). 쉘은 효률을 높이기 위해 매번 경로를 탐색하지 않고 하쉬표를 사용하여 지령행에 입력된 지령들을 찾는다. 만일 탐색경로에 이미 있던 등록부들중 어느 하나에 새 지령이 추가되면 내부하쉬표를 다시 작성해야 한다. 이것은 다음의것을 입력하여 수행한다.

% rehash
(%는 C쉘재촉문이다.)

재촉문에서 경로를 변화시키거나 또 다른 쉘을 파생할 때 하쉬표도 자동적으로 다시 작성된다.

hashstat지령 hashstat지령은 성능통계를 현시하여 하쉬표로부터 지령을 탐색하는 효률을 보여 준다. 통계는 《성공》과 《실패》로 되여 있다. 만일 쉘이 사용된 대부분의 지령들을 경로의 끝에서 찾는다면 그것들이 경로의 앞에 있을 때보다 찾기가 더 힘들어 성공보다. 실수를 더 많이 한다. 이러한 경우 성공하기가 제일 힘든 등록부를 경로의 앞에 놓아 성능을 개선할수 있다.

% hashstat
2 hits,13misses, 13%

source지령 source지령은 쉘의 내부지령 즉 쉘의 내부코드부분이다. 이것을 사용하여 파일로부터 지령이나 지령들의 모임을 실행시킨다. 보통 지령이 실행될 때 쉘은 자식프로쎄스들을 파생하여 지령을 실행시킴으로써 임의의 변화가 생겨도 그것이 어미쉘이라고 하는 초기쉘에 영향을 미치지 않도록 한다. 그러나 source지령은 프로그람이 현재쉘에서 실행되게 하여 그 파일에서 설정된 모든 변수들이 현재쉘환경의 부분으로 되게 한다. source지령은 보통 .cshrc나 .login이 다 수정되였을 때 그것들을 재실행시키기 위해 사용된다.
실례로 등록가입후에 경로가 변화되면 다음과 같이 입력한다.

% source .login or .cshrc

쉘재촉문 C쉘에는 2개의 재촉문 즉 1차재촉문(%)과 2차재촉문(?)이 있다. 1차재촉문은 등록가입한 다음에 말단에 현시되는것이다. 이것은 지령입력을 대기한다. 1차재촉문은 재설정될수 있다. 만일 이 재촉문에서 C쉘프로그람작성지령들 실례로 결심채택이나 순환을 요구하는 스크립트를 작성하고 있다면 2차재촉문이 나타나 다음행으로 계속할수 있게 한다. 이것은 그 지령이 완성될 때까지 새행으로 넘어갈 때마다 계속 나타난다. 2차재촉문은 재설정될수 없다.

1차재촉문 대화형으로 실행될 때 재촉문은 지령을 입력하고 Enter건을 누르는것

을 기다린다. 만일 기정재촉문을 사용하는것을 원하지 않으면 .cshrc파일에서 그것을 재설정한다. 그러면 그것은 이 쉘과 다른 모든 C보조쉘들에 의해 설정된다. 다만 그것을 등록가입대화조종을 위해 설정하려면 쉘재촉문에서 설정해야 한다.

실례 9-3

1. % **set prompt="$LOGNAME>"**
2. **ellie >**

설명

1. 1차재촉문에 사용자의 등록가입이름과 >부호 그리고 한개의 공백이 대입된다.
2. 새 재촉문이 현시된다.

2차재촉문 2차재촉문은 재촉문에서 즉시스크립트를 작성할 때 나타난다.

쉘프로그람작성지령들이 입력되여 새행으로 넘어 갈 때마다 2차재촉문이 나타나고 그 지령들이 완전히 끝날 때까지 계속 현시된다. 재촉문에서 스크립트들을 정확히 작성하자면 많이 해보아야 한다. 일단 지령이 입력되면 Enter건을 눌러도 그것을 예비복사할수 없으며 C쉘리력기구는 2차재촉문에서 입력된 지령들을 기억하지 않는다.

실례 9-4

1. % foreach pal (Joe torn aim)
2. **?** mail $pal < memo
3. **?** end
4. %

설명

1. 이것은 즉시스크립트작성실례이다. C쉘이 foreach순환이 입력된 다음에 입력을 더 요구하기때문에 2차재촉문이 나타난다. foreach순환은 팔호안의 목록에 있는 매 단어를 처리한다.
2. 첫번째 순환때 joe가 변수 pa1에 대입된다. 사용자 joe에 우편에 있는 memo의 내용들을 전송한다. 다음번에 순환할 때 tom이 변수 pal에 대입되고 이런 식으로 계속된다.
3. end명령은 순환의 끝을 현시한다. 팔호안의 목록에 있는 모든 항들이 처리되였을 때 순환이 끝나고 1차재촉문이 현시된다.
4. 1차재촉문이 현시된다.

3. 지령행

C쉘은 등록가입후 1차재촉문을 기정값 %로 현시한다.

쉘은 지령해석프로그람이다. 쉘이 대화형으로 실행할 때 말단으로부터 지령들을 읽어 들이고 그 지령행을 단어들로 나눈다. 지령행은 한개이상의 단어들 (혹은 통표들)로 이루어 지는데 이것들은 공백(빈 칸이나 타브들)에 의해 분리되고 Enter건을 누를 때 생기는 행바꾸기로 끝난다.

첫번째 단어는 지령이고 그다음 단어들은 지령의 추가선택들과 인수들이다.

지령은 ls나 pwd와 같은 UNIX실행프로그람일수 있고 cd나 jobs와 같은 내부지령일수도 있으며 쉘스크립트일수도 있다. 지령들은 쉘이 지령행을 구분할 때 해석해야 하는 메타문자라고 하는 특수문자들을 포함할수 있다. 만일 지령행의 마지막문자가 행바꾸기이고 그앞에 거꿀빗선이 있으면 그 행은 다음행으로 계속 될수 있다.[2]

탈퇴상태 지령이나 프로그람은 끝날 때 어미프로쎄스에 탈퇴상태를 되돌린다. 탈퇴상태는 0과 255사이의 수이다. 습관상 프로그람이 끝날 때 복귀된 상태가 령이면 그 지령은 실행에서 성공한것이다. 탈퇴상태가 령이 아니면 그 지령은 실패한것이다.

C쉘상태변수는 실행된 마지막지령의 탈퇴상태값으로 설정된다. 프로그람이 성공하는가 실패하는가는 그 프로그람을 작성하는 작성자에 의해 결정된다.

실례 9-5

1. % grep "el lie" /etc passwd
   ellie:GgMyBsSJavdl6s:9496:40:Ellie Quigley:/home/jody/ellie
2. % **echo $status**
   0
3. % grep "nicky" /etc passwd
4. % **echo $status**
   1
5. % grep "scott" /etc/passsswd
   grep: /etc/passsswd: No such file or directory
6. % **echo $status**
   2

설명

1. grep프로그람은 /etc/passwd파일에서 패턴 ellie를 탐색하여 성공한다. /etc/passwd로부터 행이 현시된다.
2. 상태변수가 grep지령의 탈퇴상태로 설정된다. 0은 성공을 표시한다.
3. grep프로그람은 사용자 nicky를 /etc/passwd파일에서 찾을수 없다.
4. grep프로그람은 패턴을 찾을수 없다. 그래서 탈퇴상태를 1로 넘긴다.
5. 파일 /etc/passwd가 열릴수 없으므로 grep가 실패한다.
6. grep가 파일을 찾을수 없으므로 탈퇴상태를 2로 되돌린다.

지령그룹화 지령행은 몇개의 지령들로 이루어 질수 있다. 매 지령은 반두점에 의해 분리되고 행바꾸기로 끝난다.

실례 9-6

% ls; **pwd;cal 2001**

---

2. 지령행의 길이는 256문자이거나 그이상이 될수 있다. UNIX의 다른 판본들에서는 그 길이가 더 길수 있다.

설명

행바꾸기가 나타날 때까지 지령들은 왼쪽에서 오른쪽으로 실행된다.
지령을 묶어서 모든 출력이 다른 지령으로 파이프되거나 파일로 방향바꾸기되게 할수 있다. 쉘은 보조쉘에서 지령들을 실행한다.

실례 9-7

1. % **(ls ; pwd; cal 2001) > outputfile**
2. % **pwd; (cd / ; pwd) ;pwd**
   /home/jody/ellie
   /
   /home/jody/ellie

설명

1. 매 지령들의 출력이 outputfile이라고 하는 파일에 전송된다. 괄호가 없으면 처음 2개의 지령들의 출력은 현시장치으로 가고 cal지령의 출력만이 출력파일로 방향바꾸기된다.
2. pwd지령은 현재작업등록부를 현시한다. 괄호는 그안에 있는 지령들이 보조쉘에 의해 처리되도록 한다. cd지령은 쉘에 내장되여 있다. 등록부는 보조쉘에서 root로 변화되고 현재작업등록부가 현시된다. 보조쉘에서 벗어 날 때 초기쉘의 현재작업등록부가 현시된다.

지령의 조건실행 조건실행에서 2개의 지령문자렬은 2개의 특수메타문자들 &&, || 에 의해 분리된다. 이 메타문자들의 오른쪽에 있는 지령은 왼쪽에 있는 지령의 탈퇴조건에 따라 실행되거나 실행되지 않는다.

실례 9-8

% **grep '^tom:' /etc/passwd && mail tom < letter**

설명

첫번째 지령이 성공하면(령탈퇴상태를 가지면) 기호 &&다음의 두번째 지령이 실행된다. grep지령이 passwd파일에서 tom을 성과적으로 찾으면 오른쪽에 있는 지령이 실행된다.
mail프로그람이 tom을 letter파일의 내용으로 보낸다.

실례 9-9

% **grep '^tom:' /etc/passwd || echo "tom is not user here."**

설명

만일 첫번째 지령이 실패하면(령이 아닌 탈퇴상태를 가지면) 표식 ||다음의 두번째 지령이 실행된다. grep지령이 passwd파일에서 tom을 찾지 못하면 오른쪽

에 있는 지령이 실행된다.
echo프로그람이 tom is not a user here를 현시장치에 인쇄한다.

배경에서 지령 보통 지령을 실행할 때 그것은 전경에서 실행되고 재촉문은 지령이 완전히 실행될 때까지 다시 나타나지 않는다. 지령이 완전히 실행될 때까지 기다리는것이 언제나 편리한것은 아니다.

마지막지령이 끝나기전에 또 다른 지령을 시작할수 있도록 쉘은 &를 지령행의 끝에 추가하여 쉘재촉문을 즉시에 복귀한다.

배경에서 실행되는 지령을 배경일감이라고 하며 출력은 그것이 처리될 때 현시장치에 전송된다. 2개의 지령들이 동시에 현시장치로 출력을 보낸다면 혼돈이 생길수 있다. 이것을 막기 위해 배경에서 실행되는 일감의 출력을 파일에 보내거나 인쇄기와 같은 다른 장치에 파이프할수 있다.

보통 배경에서 새로운 쉘창문을 시작하는것이 편리하다. 이렇게 하면 처음에 시작한 창문과 새로운 쉘창문에 다 접근할수 있다.

실례 9-10

1. % **man xview | lp&**
2. *[1] 1557*
3. %

설명

1. xview프로그람을 위한 사용지도서페지로부터의 출력이 인쇄기에 파이프된다. 지령행의 끝에 &가 있으므로 일감이 배경에 놓인다.
2. 현시장치에 2개의 수가 나타난다. 중괄호안에 있는 수는 배경에 놓이는 첫번째 파제라는것을 현시하고 두번째 수는 이 파제의 PID이다.
3. 쉘재촉문이 즉시에 나타난다. 프로그람이 배경에서 실행될 때 쉘은 전경에서 또 다른 지령을 재촉한다.

지령을 묶어서 모든 출력을 다른 지령으로 파이프하거나 파일로 방향바꾸기되게 할수 있다. 쉘은 보조쉘에서 지령들을 실행한다.

4. 지령행리력

리력기구는 C쉘에 내장되여 있다. 그것은 지령행에 입력된 지령들(리력사건들)의 번호가 붙은 목록을 기억한다. 리력목록으로부터 지령을 다시 호출하여 지령을 다시 입력하지 않고도 재실행할수 있다.

리력대입문자인 감탄부호를 흔히 bang문자라고 부른다. 내부지령 history지령은 리력목록을 현시한다.

실례 9-11

(The Command Line)
% **history**

1. *cd*
2. *ls*
3. *more /etc/fstab*
4. */etc/mount*
5. *sort index*
6. *vi index*

설명

리력목록은 지령행에 입력된 마지막지령들을 현시한다.
목록에 있는 매 사건앞에 번호가 붙는다.

리력설정 C쉘의 history변수는 리력목록으로부터 기억하고 현시장치에 현시하려고 하는 사건들의 수로 설정된다. 보통 이것은 사용자초기화파일인 .cshrc파일에서 설정된다.

실례 9-12

set history=50

설명

말단에 최근에 입력된 50개의 지령들이 보존되여 있으므로 history지령을 입력하여 현시장치에 현시할수 있다.

리력보관 등록가입할 때 그전의 리력사건들을 보존하기 위해서 savehist변수를 설정한다. 이 변수는 보통 사용자초기화파일인 .cshrc파일에서 설정된다.

실례 9-13

set savehist=25

설명

리력목록으로부터 마지막 25개 지령들이 기억되여 다음번 등록가입할 때 리력목록의 처음에 놓인다.

리력현시 history지령은 리력목록에 있는 사건들을 현시한다.

history지령은 또한 사건들의 번호와 사건들이 현시되는 형식을 조종하는 추가선택들을 가지고 있다. 사건들에 번호를 붙일 때 반드시 1부터 시작하지 않아도 된다. 만일 리력목록에 100개의 지령이 있고 history변수를 25로 설정했다면 기억된 마지막 25개의 지령들만 볼수 있다.

실례 9-14

% history
1. *ls*
2. *vi filel*
3. *df*

4. *ps -eaf*
5. *history*
6. *more /ekc/passwd*
7. *cd*
8. *echo $VSER*
9. *set*

설명

리력목록이 현시된다. 매 지령에 번호가 붙는다.

실례 9-15

% **history** *# Print without line numbers*
ls
vi file1
df
ps -eaf
history
more /etc/passwd
cd
echo $USER
set
history -n

설명

리력목록이 행번호가 없이 현시된다.

실례 9-16

% **hisrory –r** *# Print the history list in reverse*
11. *history –r*
10. *history –h*
9. *set*
8. *echo $USER*
7. *cd*
6. *more /etc/passwd*
5. *history*
4. *ps –eaf*
3. *df*
2. *vi file1*
1. *ls*

설명

리력목록이 반대순서로 현시된다.

실례 9-17

```
% history 5          # Print the last 5 events on the history list
 7.  echo $USER
 8.  cd
 9.  set
10. history –n
11. history 5
```

설명

리력목록의 마지막 5개 지령이 현시된다.

지령재실행 리력목록으로부터 지령을 재실행하기 위하여 감탄부호를 사용한다. 만일 2개의 감탄부호를 사용하면 마지막지령이 재실행된다. 감탄부호다음에 한개 수자를 입력하면 그 수자가 리력목록의 지령과 련관되여 그 번호의 지령이 실행된다.

감탄부호와 한개의 문자를 입력하면 그 문자로 시작되는 제일 마지막지령이 실행된다. 탈자기호(ˆ)도 이전 지령을 편집하기 위한 지름방법으로 사용된다.

실례 9-18

```
1.  % date
    Mon Feh   8 12:27:35 PST 2001
2.  %  !!
    date Won Feb   8 12:28.25 PST 2001
3.  % !3
    date Mon Feb   8 12:29:26 PST 2001
4.  % !d
    date Mon Feb   8 12:30:09 PST 2001
5.  % dare
    dare: Command not Sound.
6.  % ^r^t
    date
    Mon Feb   8 12:33:25 PST 2001
```

설명

1. UNIX의 date지령이 지령행에서 실행된다. 리력목록이 갱신된다. 이것이 목록에 제일 마지막으로 놓이는 지령이다.
2. !!는 리력목록으로부터 제일 마지막지령이다. 그 지령이 재실행된다.
3. 리력목록에 있는 세번째 지령이 재실행된다.
4. 문자 d로 시작되는 리력목록의 제일 마지막지령이 재실행된다.
5. 지령이 잘못 입력된다.

6. 탈자기호를 사용하여 리력목록의 마지막지령으로부터 문자들을 대입한다. 첫번째로 나타난 r가 t로 바뀌여 진다.

실례 9-19

1. % **cat file1 file2 file3**

   <Contents of files displayed here>

   % **vi !:1**
   vi file1

2. % **cat file1 file2 file3**

   <Contents of file1, file2, and file3 are displayed here>

   % **ls !:2**
   ls file2
   file2

3. % **cat file1 file2 file3**
   % **ls !:3**
   ls file3
   file3

4. % **echo a b c**
   a b c
   % **echo !$**
   echo c
   c

5. % **echo a b c**
   a b c
   % **echo !^**
   echo a
   a

6. % **echo a b c**
   a b c
   % **echo !***
   echo a b c
   a b c

7. % **!!:p**
   echo a b c

설명

1. cat지령은 file1의 내용들을 현시장치에 인쇄한다. 리력목록이 갱신된다. 지령행은 단어번호가 0으로부터 시작되는 단어들로 나누어 진다. 단어번호앞에 옹근두점이 있으면 그 단어는 리력목록으로부터 추출될수 있다. !:1표기는 리력목록의 마지막지령으로부터 첫번째 인수를 취하여 그것을 지령문자렬에 다시 넣으라는 뜻이다. 마지막지령의 첫번째 인수는 file1이다(단어

0은 지령 그자체이다.).
2. !:2가 마지막지령의 두번째 인수 file2로 교체되고 ls에 인수로서 주어 진다. file2가 인쇄된다(file2는 3번째 단어이다.).
3. ls!:3은 리력목록의 마지막지령으로 가서 네번째 단어 (단어들은 0에서 시작한다.)를 취하여 그것을 ls지령에 인수로서 넘긴다(file3은 네번째 단어이다.).
4. 감탄부호(!)는 표식 ($)과 함께 리력목록에서 마지막지령의 마지막인수를 나타낸다. 마지막인수는 c이다.
5. 탈자기호(^)는 지령다음의 첫번째 인수를 표시한다. 감탄부호(!)는 탈자기호와 함께 리력목록에서 마지막지령의 첫번째 인수를 가리킨다. 마지막지령의 첫번째 인수는 a이다.
6. 별표(*)는 지령뒤에 있는 모든 인수들을 표시한다. 감탄부호(!)는 별표와 함께 리력목록에 있는 마지막지령의 모든 인수들을 표시한다.
7. 리력목록의 마지막지령이 인쇄되고 실행되지는 않는다. 리력목록이 갱신된다. 행에서 탈자기호대입을 할수 없다.

5. 별명

별명은 지령에 대한 C쉘의 사용자정의된 략자이다. 지령이 많은 추가선택들과 인수들을 가지고 있거나 문법을 기억하기 힘들 때 별명을 쓰는것이 좋다. 지령행에서 설정된 별명들은 보조쉘들에 의해 계승되지 않는다. 별명들은 보통 .cshrc파일에서 설정된다. .cshrc는 새로운 쉘이 시작될 때 실행되므로 거기서 설정된 임의의 별명들은 새로운 쉘을 위해 재설정된다. 별명들은 쉘스크립트들에 넘겨 질수도 있지만 그것들이 스크립트안에서 직접 설정되지 않으면 이식성문제를 초래할수 있다.

별명목록제시 내부지령 alias는 설정된 모든 별명들의 목록을 현시한다. 별명이 먼저 인쇄되고 그다음에 그것이 표시하는 실지 지령이나 지령들이 인쇄된다.

실례 9-20

```
% alias
co              compress
cp              cp -i
lsl     enscripk -B -r -Porange -f Couriers !*
mailq    /usr/lib/sendinail -bp
more     more
mv       iav -i
rn       /usr/spool/news/bin/rn.3
uc       uncompress
uu       uudecode
vg      vgrind –t –s11 !:1 | lpr –t
weekly (cd /home/jody/ellie/activity; ./weekly_report; echo Done)
```

설명

alias지령은 첫번째 렬에 지령에 대한 별명(가명)을 현시하고 두번째 렬에 그

별명이 나타내는 실지지령을 현시한다.

별명만들기 alias지령을 사용하여 별명을 만든다. 첫번째 인수는 별명의 이름 즉 지령에 대한 가명이다. 그 행의 나머지 부분은 별명이 실행될 때 실행되는 한개의 지령이나 지령들로 이루어 진다. 여러개의 지령들은 반두점에 의해 구분되고 공백과 메타문자들이 있는 지령들은 단일인용부호안에 넣는다.

실례 9-21

1. % **alias m more**
2. % **alias mroe more**
3. % **alias lf 'ls –alF'**
4. % **alias cd 'cd \ !*; set prompt = "$cwd >"'**

   % **cd ..**

   /home/jody **>** ***cd /*** *# New prompt displayed*

   / >

설명

1. more지령에 대한 가명이 m으로 설정된다.
2. more지령에 대한 별명이 mroe로 설정된다. 이렇게 하면 글자를 모를 때 편리하다.
3. 별명정의가 공백때문에 인용부호안에 놓인다. 별명 lF는 지령 ls-alF에 대한 가명이다.
4. cd가 실행될 때 그에 대한 별명은 cd를 인수로서 명기된 등록부로 넘기고 재촉문을 뒤에 문자렬 >이 붙은 현재작업등록부로 재설정한다. !*는 리력기구에 의해 사용되는것과 같은 방법으로 별명에 의해 사용된다. 거꿀빗선은 별명이 !*를 사용하기전에 먼저 리력기구에 의해 사용되지 않게 한다. !*는 리력목록에 있는 가장 최근의 지령으로부터 인수들을 현시한다.

별명지우기 unalias지령을 사용하여 별명을 지운다. 림시로 별명을 사용하지 않으려면 별명이름앞에 거꿀빗선을 붙인다.

실례 9-22

1. % **unalias more**
2. % \ **cd..**

설명

1. unalias지령은 정의된 별명들의 목록으로부터 별명 mroe를 지운다.
2. 별명 cd는 이 지령의 실행에 대해서만 쓸수 없게 된다.

별명순환 별명정의가 초기별명을 다시 참조하는 또 다른 별명을 참조할 때 별명순환이 일어 난다.

실례 9-23

```
1. % alias m more
2. % alias mroe m
3. % alias m mroe                    # Cauases a loop
4. % m datafile
   Alias loop.
```

설명

1. m은 별명이다. more는 별명정의이다. m이 사용될 때마다 more지령이 실행된다.
2. More는 별명이다. M은 별명정의이다. More가 입력되면 별명 m이 호출되여 more지령이 실행된다.
3. 이것은 아무것도 하지 않는다. 별명 m이 사용되면 그것은 별명 mroe를 호출하고 별명 mroe는 다시 m을 참조하여 별명을 순환시킨다. 그 어떤 부정적결과도 생기지 않는다. 다만 오유통보가 나타난다.
4. 별명 m이 사용된다. 그것은 순환된다. 처음에 m은 mroe를 호출하고 mroe는 m을, 그다음에 m은 mroe를 호출하는 식으로 계속된다. C쉘은 그것을 포착하여 별명을 무한히 순환시키지 않고 오유통보를 현시한다.

6. 일감조종

일감조종은 jobs라고 하는 프로그람들을 배경이나전경에서 실행할수 있게 하는 C쉘의 강력한 수단이다. 보통 지령행에 입력된 지령은 전경에서 실행되여 그것이 끝날 때까지 계속된다. 만일 여러개의 창문이 있다면 과제조종은 필요 없게 된다. 왜냐하면 이때 새로운 일감을 시작하기 위해 다른 창문을 열기만 하면 되기때문이다. 그러나 말단이 하나일 때 일감조종은 매우 유용한 수단이다. 표 9-1에 일감지령들의 목록을 보여주었다.

표 9-1. 일감조종지령

지 령	의 미
Jobs	실행되는 모든 일감들을 현시한다.
^Z(Control-Z)	일감을 정지(중지)시킨다. 재촉문이 현시장치에 나타난다.
bg	배경에서 중지된 일감실행을 시작한다.
fg	배경일감을 전경으로 가져 간다.
kill	중지신호를 지정된 일감에 보낸다.

앰페샌드(&)와 배경일감 만일 지령이 오래동안 실행되면 그 지령에 &기호를 추가하여 그 일감을 배경에서 실행시킬수 있다. 이때 C쉘재촉문이 즉시에 나타나 또 다른 지령을 입력할수 있다. 그러면 2개의 지령이 동시에 실행된다. 하나는 배경에서, 다른 하나는 전경에서 진행된다. 이것들은 다 자기의 표준출력을 현시장치으로 보낸다. 따라서 일감을 배경에 놓았을 때 그 출력을 현시장치에 방향바꾸기하거나 인쇄기와 같은 장치에 파이프하는것이 좋다.

실례 9-24

1. % **find . –name core –exec rm {} \ ; &**
2. *[1] 543*
3. %

설명

1. find지령은 배경에서 실행된다(-print 추가선택이 없으면 find지령은 어떤 출력도 현시장치에 보내지 않는다.).[3]
2. 중괄호안의수는 이것이 배경에서 실행되는 첫번째 일감이라는것을 현시한다. 543은 프로쎄스의 PID이다.
3. 재촉문이 즉시에 복귀한다. 쉘이 사용자입력을 대기한다.

중지건조종문자와 배경일감 프로그람을 중지시키기 위하여 중지건조종문자인 ^Z를 설정한다. 일감이 중지(정지)되면 쉘재촉문이 현시되고 fg나 bg지령이 설정될 때까지 그 프로그람은 다시 실행하지 않는다(vi편집기를 사용할 때 ZZ지령은 파일을 쓰고 기억한다. 이것을 vi대화조종을 중지하는 ^Z와 혼돈하지 말아야 한다.).

만일 일감이 중지될 때 등록탈퇴하려고 하면 통보 There are stopped jobs가 현시장치에 나타난다.

jobs지령 C쉘의 내부지령 jobs는 현재 실행되고 있는 프로그람들과 배경에서 실행되거나 중지된 프로그람들을 현시한다. 일감이 정지될 때 그것은 중지된다. 즉 실행상태에 있지 않다. 이 두 경우에 말단은 다른 지령들을 접수할수 있다.

실례 9-25

(The Command Line)
1. % **jobs**
2. *[1] + Stopped vi filex*
   [2] – Running sleep 25
3. % jobs –1
   [1] + 355 Stopped vi filex
   [2] – 356 Running sleep 25
4. *[2] Done sleep 25*

설명

1. jobs지령은 현재 진행되는 일감들을 현시한다.
2. 표기 [1]은 첫번째 일감의 번호이다. 더하기부호는 이 일감이 배경에 제일 마지막에 들어 간 일감이 아니라는것을 표시한다. 횡선은 이것이 배경에 제일 마지막에 놓인 일감이라는것을 현시한다. 정지되였다라는 말은 이 일감이 ^Z건으로 중지되여 지금 진행되지 않는다는것을 의미한다.

---

3. find 문법에서 exec 명령의 끝에 반두점이 있어야 한다. 반두점앞에 거꿀빗선을 붙이면 쉘이 그것을 해석하지 않게 한다.

3. -l추가선택(긴표시)는 과제의 번호뿐만이 아니라 PID로 현시한다. 표기 [2]는 두번째 과제의 번호인데 이 경우에는 배경에 넣어 진 마지막일감이다. 횡선은 이것이 제일 최근의 일감이라는것을 현시한다. sleep지령이 배경에서 실행되고 있다.
4. sleep지령이 25s동안 실행된 다음에 과제는 끝나고 그것이 끝났다는것을 알려 주는 통보가 현시장치에 나타난다.

전경과 배경지령 fg지령은 배경일감을 전경에로 넘긴다. bg지령은 배경에서 중지된 일감을 시작한다. 만일 일감조종을 위하여 어떤 특수일감을 선택하려면 %표식과 그 일감의 번호를 fg와 bg에 대한 인수로 사용할수 있다.

실례 9-26

1. % **jobs**
2. *[1] + Stopped vi filex*
   [2] – Running cc prog.c -o prog
3. % fg %1
   vi filex
   (vi session starts)
4. % kill %2
   [2] Terminated cc prog.c -o prog
5. % **sleep 15**
   (Press ^z)
   Stopped
6. % **bg**
   [1]   sleep 15&
   [1] Done sleep 15&

설명

1. jobs지령은 jobs라고 하는 현재 실행되는 프로쎄스들을 현시한다.
2. 정지된 첫번째 일감은 vi대화조종이고 두번째 일감은 cc지령이다.
3. 번호가 [1]인 과제가 전경으로 넘어 간다. 이 번호앞에 %표식이 있다.
4. kill지령이 내부지령이다. 그것은 프로쎄스에 기정으로 TERM(끌)신호를 보낸다. 인수는 번호나 프로쎄스의 PID이다.
5. sleep지령이 ^Z를 누를 때 정지된다. sleep지령은 CPU를 사용하고 있지 않으며 배경에서 중지된다.
6. bg지령은 마지막배경일감이 배경에서 실행을 시작하도록 한다. sleep지령은 실행이 다시 시작되기전에 초로 내리계수를 시작한다.[4]

---

[4]. grep, sed, awk와 같은 프로그람들은 패턴일치을 위해 정규표현메타문자라고 하는 메타문자들의 모임을 가지고 있다. 이것들을 쉘메타문자들과 혼돈하지 말아야 한다.

7. 메타문자

메타문자들은 자기자체가 아닌 다른것을 표시하는데 사용되는 특수문자들이다. 실천경험으로 보면 자모문자도 아니고 수자도 아닌 문자들이 메타문자들이다. 쉘은 흔히 쉘통용기호라고 하는 메타문자들의 모임을 가지고 있다.

쉘메타문자들을 사용하여 지령묶기, 파일이름과 경로이름의 간략화, 입출력을 방향바꾸기, 지령들을 배경에 넣기 등 여러가지 조작을 할수 있다. 표 9-2는 쉘메타문자들의 부분목록을 보여 준다.

표 9-2. 쉘메타문자

메타문자	목 적	실 례	의 미
$	변수바꿔넣기	setname=Tom echo $name Tom	변수 name을 Tom으로 설정한다. 거기에 기억된 값을 현시한다.
!	리력바꿔넣기	!3	리력목록으로부터 세번째 사건을 재실행한다.
*	파일이름바꿔넣기	Rm*	모든 파일들을 지운다.
?	파일이름바꿔넣기	ls??	2개문자로 된 모든 파일들을 지운다.
[]	파일이름바꿔넣기	catf[123]	f1, f2, f3의 내용들을 현시한다.
;	지령분리기	ls;date;pwd	매 지령이 차례로 실행된다.
&	배경처리	lp mbox&	배경에서 인쇄가 진행된다. 재촉문이 즉시에 복귀한다.
>	출력방향바꾸기	ls>file	표준출력을 파일로 방향바꾸기한다.
<	입력방향바꾸기	ls<file	표준입력을 파일로부터 방향바꾸기한다.
>&	출력과 오유방향바꾸기	ls>&file	출력과 오유를 다 파일로 방향바꾸기한다.
>!	noclobbr가 설정되면 그것을 무시한다.	ls>!file	파일이 존재하며 noclobber가 설정되여도 그 파일의 끝을 자르고 우에 덧쓴다.
>>!	noclobber가 설정되면 그것을 무시한다.	ls>>!file	파일이 존재하지 않으면 noclobber가 설정되여도 그 파일을 작성한다.
()	보조쉘에서 실행될지령들을 묶는다.	(ls;pwd)>tmp	지령들을 실행하고 출력을 tmp파일로 보낸다.
{}	이 쉘에서 실행될지령들을 묶는다.	{cd/;echo $cwd}	Root등록부로 변화시키고 현재작업등록부를 현시한다.

파일이름바꿔넣기 쉘은 지령행을 평가할 때 문자들의 어떤 모임과 대응하는 메타문자들을 사용하여 파일이름이나 경로이름들을 간략화한다. 표 9-3에서 보여 준 파일이름꿔넣기메타문자들은 자모문자로된 파일이름들의 모임으로 확장된다. 메타문자들을 파일이름으로 확장하는 과정을 파일이름확장이라고도 한다. 다른 쉘들과 달리 C쉘은 파

일이름으로써 그것을 표시하는 메타문자를 대신할수 없으면 No match를 통보한다.

표 9-3. 쉘메타문자와 파일이름대입

메타문자	의 미
*	임의의 수의 문자들을 대신한다.
?	한개 문자만을 대신한다.
[abc]	문자모임에서 한개문자 즉 a, b 또는 c를 대신한다.
[a-z]	a부터 z사이의한개문자를 대신한다.
{a,ile,ax}	한개 문자나 문자들의 모임을 대신한다.
~	사용자의 뿌리등록부로 물결표 ~를 대신한다.
\	메타문자를 쓸수 없게 한다.

8. 메타문자확장

쉘은 자기의 메타문자들을 평가하고 그것들을 파일이름의 적당한 문자나 수자로 교체하여 파일이름바꿔넣기를 진행한다.

별표 별표는 파일이름에서 임의의 수의 문자들을 대신한다.

실례 9-27

1. % **ls**
   a.c h.c abc ab3 filel file2 file3 file4 file5

2. % **echo ***
   a.c b.c abc ab3 filel file2 file3 file4 file5

3. % ls ***.c**
   a.c b.c

4. % **rm z*p**
   No match.

설명

1. 현재등록부에 있는 모든 파일들이 현시된다.
2. echo프로그람은 모든 인수들을 현시장치에 인쇄한다. 별표는 파일이름에 있는 임의의 수의 문자들을 대신하는 통용기호이다. 등록부에 있는 모든 파 일들이 일치되여 현시장치에 출력된다.
3. c로 끝나는 파일이름들이 현시된다.
4. 등록부에 z로 시작하는 파일이 없으므로 쉘은 No match를 통보한다.

물음표 물음표는 파일이름에서 한개문자만을 대신한다.

실례 9-28

1. % **ls**
   a.c b.c abc ab3 filel file2 file3 file4 file5
2. % **ls ???**
   abc ab3
3. % **echo How are you?**
   No match.
4. % **echo How are you\ ?**
   How are you?

설명

1. 현재등록부에 있는 모든 파일들이 현시된다.
2. 물음표는 파일이름에서 한개문자를 대신한다. 3개 문자들로 된 모든 파일이름들이 현시된다.
3. 쉘은 you뒤에 한개문자가 더 있는 파일이름을 찾는다. 등록부에 이 문자들과 맞는 파일이 하나도 없다. 쉘은 No match를 인쇄한다.
4. 물음표앞에 거꿀빗선을 사용하여 물음표가 특수한 의미로 쓰이지 않게 한다. 따라서 쉘은 물음표를 문자자체로 취급한다.

중괄호 중괄호는 문자들의 모임이나 렬로부터 한개문자를 파일이름으로 대신한다.

실례 9-29

1. % **ls**

   a.c b.c abc ab3 filel file2 file3 file4 file5 file10 filell filel2
2. % **ls file[123]**

   filel file2 file3
3. % **ls [A-Za-z]la-z][1-5]**

   ab3
4. % **ls file1 [0-2]**

   file10 filell filel2

설명

1. 현재등록부에 있는 모든 파일들이 현시된다.
2. file로 시작되여 그뒤에 1, 2, 3중에서 어느 하나가 붙은 파일이름들이 일치되여 현시된다.
3. 한개 문자(대문자나 소문자)로 시작되고 그다음에 한개 소문자와 1부터 5사이의 한개 수자가 오는 파일이름들이 일치되여 현시된다.
4. file1로 시작되고 0, 1, 2중에서 어느 하나가 놓이는 파일이름들이 현시된다.

대괄호 대괄호({})는 파일이름에서 한개문자나 문자들의 렬을 대신한다.

실례 9-30

1. % **ls**
   a.c b.c abc ab3 ab4 ab5 file1 file2 file3 file4 file5 foo faa fumble
2. **ls f{oo,aa,umble}**
   foo faa fumble
3. % **ls a{.c,c,b[3-5]}**
   a.c ab3 ab4 ab5

설명

1. 현재등록부에 있는 모든 파일들이 현시된다.
2. f로 시작되고 그다음에 문자렬들인 oo나 aa 또는 umble이 오는 파일들이 현시된다. 대괄호안에 공백이 있으면 오유통보 Missing}가 나타난다.
3. a로 시작되여 z뒤에 .c, c, b3, b4 또는 b5가 오는 파일들이 현시된다(중괄호는 대괄호안에서 사용할수 있다.).

메타문자에서 벗어 나기 거꿀빗선을 사용하여 문자가 특수한 의미에서 벗어 나게 한다. 벗어 난 문자는 그자체를 현시한다.

실례 9-31

1. % **gotta light?**
   No match
2. % **gotta light\ ?**
   gotta: Command not found.

설명

1. 이 행은 UNIX를 한번 시험해 본다. 물음표는 파일바꿔넣기메타문자이므로 한개의 문자와 같다. 쉘은 현재작업등록부에서 l-i-g-h-t로 시작되고 그뒤에 한개문자가 오는 파일을 찾는다. 쉘이 파일을 찾지 못하면 No match를 통보한다. 이것을 통해 쉘이 지령행을 구분하는 순서를 알수 있다. 메타문자는 쉘이 gotta지령을 찾기전에 평가된다.
2. 거꿀빗선은 메타문자가 해석되지 않게 하는데 이것을 보통 메타문자에서 벗어 나기라고 한다. 이때 쉘은 No match를 통보하지 않고 gotta지령에 대한 경로를 탐색하는데 그 지령은 발견되지 않는다.

물결표확장 물결표기호는 단독으로 사용자의 홈등록부의 완전경로이름으로 확장된다. 물결표가 사용자이름앞에 붙으면 그것은 그 사용자의 홈등록부의 완전경로이름으로 확장된다. 경로앞에 붙으면 홈등록부와 경로이름의 나머지 부분으로 확장된다.

실례 9-32

1. % **echo ~**

/home/jody/ellie

2. % **cd -/desktop/perlstuff**
   % **pwd**
   /home/jody/ellie/desktop/perlstuff

3. % **cd ~joe**
   % **pwd**
   /home/bambi/joe

설명

1. 물결표는 사용자의 홈등록부로 확장된다.
2. 경로이름앞에 있는 물결표는 사용자의 홈등록부와 그뒤에 오는 /desktop/perlstuff로 확장된다.
3. 사용자이름앞에 붙은 물결표는 사용자의 홈등록부로 확장된다. 이 실례에서 등록부는 사용자의 홈등록부로 변경된다.

파일이름완성: filec변수 C쉘은 대화형으로 실행할 때 파일이름이나 사용자이름을 입력하기 위한 지름방식을 제공한다. 내부변수 filec를 설정하여 파일이름을 완성한다. 현재작업등록부에서 파일의 중요한 첫번째 몇개문자들을 입력하고 ESC건을 누르면 쉘은 같은 문자들로 시작하는 파일들이 많지 않을 때 그 파일이름의 나머지부분을 채운다.

파일을 부분적으로 입력하고 Control-D를 입력하면 쉘은 그 문자들과 대응하는 파일들의 목록을 인쇄한다. 만일 대응되는것들이 많으면 말단은 경보를 울린다. 목록이 물결표로 시작하면 쉘은 그 목록을 사용자이름으로 확장하려고 한다.

실례 9-33

1. % **set filec**
2. % ls
   rum rumple rumplestilsken run2
3. % **ls ru[ESC]**[5] *# terminal beeps*
4. % **ls rum^D**
   rum 2-LL-npie ruiaplestilsken
5. % **ls rump[ESC]**
   rumple
6. % echo ~ell[ESC]
   /home/jody/ellie

설명

1. C쉘의 특수변수 filec가 설정된다. 파일이름완성을 사용할수 있다.
2. 현재작업등록부에 있는 목록들이 현시된다.
3. 파일이름완성을 시도한다. 문자 r와 u는 유일하지 않다. 즉 쉘은 어느것을 선택해야 하는지 알수 없으므로 말단이 경보를 울린다.

---

5. [ESC]는 Escape건을 의미한다.

4. 문자 r, u, m이 입력된 다음 ^D가 눌리워 진다. rum으로 시작하는 모든 파일이름들의 목록이 현시된다.
5. rump로 시작하는 첫번째 파일이름이 완성되여 현시된다.
6. 부분적으로 씌여 진 사용자이름앞에 물결표가 붙으면 쉘은 사용자이름의 문자를 완성하고 사용자홈등록부를 현시한다.

noglob를 사용한 메타문자지우기 만일 noglob가 설정되면 모든 메타문자들이 그자체를 나타내고 파일이름대입을 할수 없다. 즉 그것들은 통용기호로 사용되지 않는다. 이것은 쉘이 확장하려고 하는 메타문자들을 포함하고 있는 grep sed 또는 awk와 같은 프로그람들에서 패턴들을 탐색할 때 쓸모가 있다.

실례 9-34

1. % **set noglob**
2. % **echo * ?? []**
   ** ?? [ ]~-*

설명

1. 변수 noglob가 설정된다. 이것은 통용기호의 특수한 의미를 없앤다.
2. 메타문자들이 해석되지 안고 그자체로 현시된다.

9. 방향바꾸기와 파이프

보통 지령으로부터 표준출력(stdout)은 현시장치으로 나가고 표준입력(stdin)은 건반으로부터 들어 오며 오유통보(stderr)는 현시장치으로 나간다. 쉘은 특수방향바꾸기 메타문자들을 사용할수 있게 하여 입력과 출력을 파일에나 파일로부터 방향바꾸기한다.

방향바꾸기연산자(<, >, >>, >)는 파일이름앞에 놓는다. 이 파일은 왼쪽에 있는 지령이 실행되기전에 쉘에 의해 열린다. 수직선기호(|)로 현시되는 파이프는 한개지령의 출력이 다른 지령의 입력으로 보내질수 있게 한다. 파이프의 왼쪽에 있는 지령은 그것이 파이프에 쓰기때문에 작성자라고 부른다. 파이프의 오른쪽에 있는 지령은 그것이 파이프로부터 읽어 들이기때문에 독자라고 부른다.

방향바꾸기와 파이프메타문자들을 표 9-4에 주었다.

표 9-4. 방향바꾸기메타문자

메타문자	의 미
command < file	파일로부터 지령으로 입력을 방향바꾸기한다.
command > file	출력을 지령으로부터 파일에로 방향바꾸기한다.
command > file	출력과 오유를 파일로 방향바꾸기한다.
command >> file	지령의 출력을 방향바꾸기하여 그것을 파일에 추가한다.
command >> file	지령의 출력과 오유를 파일에로 방향바꾸기하고 추가한다.

(표계속)

메타문자	의 미
<input>	여기서 사용자입력이 진행된다. 이것은 2중괄호안에 있는 본문 문자렬로 취급된다.
WORD	WORD는 지령에 대한 입력의 끝을 현시한다.
Command | command	첫번째 지령의 출력을 두번째 지령의 입력으로 파이프한다.
Command | command	첫번째 지령의 출력은 오유를 두번째 지령의 입력으로 파이프한다.
command >! File	만일 noclobber변수가 설정되면 이 지령에 대한 그 영향을 무시하고 파일을 열거나 파일우에 덧쓴다.
command >>! File	noclobber 변수를 무시한다. 만일 파일이 존재하지 않으면 그것이 작성되고 지령의 출력이 그것에 추가된다.
command >> ! file	noclobber 변수를 무시한다. 만일 파일이 존재하지 않으면 그것이 작성되고 출력과 오유가 둘다 그것에 추가된다.

입력방향바꾸기 입력을 말단건반으로부터 들어 오게 하지 않고 파일로부터 방향바꾸기되게 할수 있다. 쉘은 < 부호의 오른쪽에 있는 파일을 열고 왼쪽에 있는 프로그람은 그 파일로부터 읽어 들인다. 만일 파일이 존재하지 않으면 오유 No such file or directory가 C쉘에 의해서 현시된다.

형식

지령 < file

실례 9-35

mail bob < memo

설명

파일 memo가 쉘에 의해 열리고 입력이 mail프로그람으로 방향바꾸기된다. mail프로그람에 의해서 사용자 bob에게 memo라고 하는 파일이 전송된다.

here문서 here문서는 입력을 지령에로 방향바꾸기하는 다른 방법이다. 차림표들을 작성하고 다른 프로그람에서 오는 입력을 처리하기 위하여 쉘스크립트에서 이것을 사용한다. 보통 건반으로부터 입력을 받는 프로그람은 Control-D(^d)로 끝난다.

here문서를 사용하여 입력을 프로그람에로 보내며 ^D를 입력하지 않고 입력을 끝낼수 있다. 기호 <<다음에 흔히 최종완료자라고 하는 사용자정의된 단어가 놓인다. 입력은 최종완료자가 나타날 때까지 기호<<의 왼쪽에 있는 지령에로 방향바꾸기된다. 최종완료자는 행에 단독으로 놓이며 그 옆에 임의의 공백도 있을수 없다. 변수와 지령바꿔넣기는 here문서안에서 진행된다(보통 here문서는 차림표를 작성하고 입력을 mail, bc, ex, ftp등과 같은 지령에 보내기 위해 쉘스크립트들에서 사용된다.).

형식

지령 << MARK

...input...

MARK

실례 9-36

(Without the here document)
(The Command Line)
1. % **cat**
2. **Hello There.**
   How are you?
   I'm tired of this.
3. **^D**

(The output)
4. *Hello There*
   How are you?
   I'm tired of this.

설명

1. 인수들이 없는 cat프로그람은 건반입력을 대기한다.
2. 사용자는 건반으로 입력한다.
3. 사용자는 ^D를 입력하여 cat프로그람에 대한 입력을 끝낸다.
4. cat프로그람은 출력을 현시장치로 보낸다.

실례 9-37

(With the *here document*)
(The Command Line)
1. % **cat << DONE**
2. **Hello There .**
   How are you?
   I'm tired of this.
3. **DONE**

(The Output)
4. *Hello There.*
   How are you?
   I'm tired of this.

설명

1. cat프로그람은 첫번째 DONE로부터 마지막 DONE까지 입력을 받는다. 이 단어들은 사용자정의된 최종완료자이다.
2. 이 행들은 입력이다. 단어 DONE이 나타날 때 입력을 그만둔다.
3. 최종완료자는 입력의 끝을 표시한다. 이 단어의 량옆에 공간이 있을수 없다.
4. 첫번째 단어 DONE과 마지막단어 DONE사이에 있는 본문이 cat지령의 출

력이며 현시장치으로 전송된다. 마지막 DONE은 행의 제일 왼쪽 끝에 있어야 하며 그 오른쪽에는 공백이나 다른 본문이 있어서는 안된다.

실례 9-38

(The Command Line)
1. % **set name = steve**
2. % **mail $name << EOF**
3. **Hello there, $name**
4. **The hour is now `date +%H`**
5. **EOF**

설명

1. 쉘변수 name에 사용자이름 steve가 대입된다.
2. 변수 name은 here문서안에서 확장된다.
3. 최종완료자 EOF가 나타날 때까지 mail프로그람은 입력을 받는다.
4. here문서안에서 지령바꿔넣기가 진행된다. 즉 거꿀인용부호안에 있는 지령이 실행되고 지령의 출력은 문자렬안에서 교체된다.
5. 최종완료자가 나타나고 mail프로그람에 대한 입력이 끝난다.

출력방향바꾸기 한개 지령이나 지령들의 표준출력은 보통 기정으로 현시장치말단으로 나간다. 현시장치로부터 표준출력을 방향바꾸기하기 위해 기호 >를 사용한다. 지령은 기호 >의 왼쪽에 있고 파일이름은 오른쪽에 있다. 쉘은 기호 >의 오른쪽에 있는 파일을 연다. 만일 파일이 존재하지 않으면 쉘은 그것을 작성한다. 파일이 존재하면 쉘은 그 파일을 열고 끝을 자른다.

보통 방향바꾸기를 사용할 때 부주의로 파일들이 지워 질수 있다(noclobber라고 하는 C쉘의 특수변수를 설정하여 방향바꾸기할 때 존재하는 파일이 파괴되지 않게 할 수 있다(표 9-6).).

형식

지령 > file

실례 9-39

cat file1 file 2 > file3

설명

file1과 file2의 내용들이 련결되여 그 출력이 file3에 보내진다. 쉘이 cat지령을 실행하기전에 file3을 연다는것을 명심해야 한다. 만일 file3이 이미 존재하고 자료를 포함하고 있다면 그 자료는 없어 진다. file3이 존재하지 않으면 그것은 작성된다.

출력을 존재하는 파일에 추가하기 출력을 존재하는 파일에 추가하기 위해 >>기호를 사용한다. 이 기호의 오른쪽에 있는 파일이 존재하지 않으면 그것이 작성된다. 만일 파일이 존재한다면 그것이 열려 지고 출력이 그 파일의 끝에 추가된다.

형식

지령 >> file

실례 9-40

date >> outfile

설명

Date지령의 표준출력이 outfile로 방향바꾸기되여 추가된다.

출력과 오유방향바꾸기 >기호를 사용하여 표준출력과 표준오유를 파일로 방향바꾸기한다. 보통 지령은 성공하면 그 출력을 표준출력으로 보내고 실패하면 그 오유통보를 표준오유로 보낸다. find나 du와 같은 일부 재귀프로그람들은 등록부나무를 거칠 때 표준출력과 표준오유를 둘다 현시장치로 보낸다.

<기호를 사용하면 표준출력과 표준오유를 다 한개 파일에 보존하고 검사할수 있다. C쉘은 표준오유만을 방향바꾸기하기 위한 기호를 제공하지 않지만 보조쉘에서 지령을 실행시켜 표준오유를 얻을수 있다(그림 9−2).

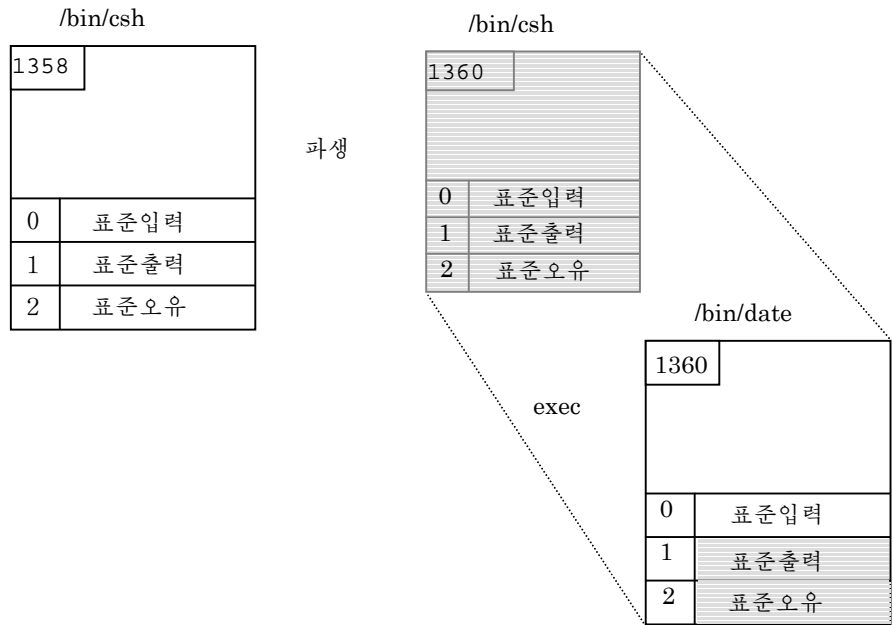



그림 9-2. 표준출력과 표준오유방향바꾸기(실례 9-41)

실례 9-41

1. % **date**
   Tue Aug 3 10:31:56 PDT 2001
2. % **date >& outfile**

3. % **cat outfile**
   Tue Aug 3 10:31:56 PDT 2001

설명

1. date지령의 출력이 표준출력인 현시장치로 보내진다.
2. 출력과 오유들이 outfile로 보내진다.
3. 오유가 없으므로 표준출력이 outfile로 보내지고 그 파일의 내용들이 현시된다.

실례 9-42

1. % **cp filel file2**
2. % **cp filel**
   Usage: cp [-ip] f1 f2; or: cp [-ipr] fl ... fn d2
3. % **cp file1 >& errorfile**
4. % **cat errorfile** *2*
   Usage: cp [-ip] f1 f2; or: cp [-ipr] f1 ... fn d2

설명

1. 파일을 복사하기 위해 cp지령은 원천파일과 목적파일을 요구한다. cp지령은 file1을 file2에 복사한다. cp지령의 문법이 정확하기때문에 현시장치에 아무것도 현시되지 않는다. 복사가 성공했다.
2. 이때 목적파일이 없으므로 cp지령이 실패하여 오유를 표준오유인 말단으로 보낸다.
3. >기호를 사용하여 표준출력과 표준오유를 둘다 errorfile에 보낸다. 지령의 유일한 출력이 오유통보이므로 이것이 errorfile에 보존된다.
4. errofile의 내용이 현시되여 그것이 cp지령에 의해 생긴 오유통보를 포함하고 있다는것을 보여 준다.

출력과 오유를 분리 표준출력과 표준입력은 지령을 괄호안에 넣어 분리시킬수 있다. 지령이 괄호안에 있으면 C쉘은 보조쉘을 기동하고 그것으로부터 방향바꾸기를 처리한 다음 지령을 실행시킨다. 실례 9-43에서 사용한 방법을 사용하여 표준출력을 오유로부터 분리시킬수 있다.

실례 9-43

(The Command Line)

1. % **find . -name '*.c' –print >& outfile**
2. % **(find . –name '*.c' -print > goodstuff) >& badstuff**

설명

1. find지령은 현재등록부에서 시작하여 .c로 끝나는 모든 파일들을 탐색하고 그 출력을 outputfile에 인쇄한다. 오유가 생기면 그것도 outputfile로 보내진다.

2. find지령이 괄호안에 있다. 쉘은 보조쉘을 작성하여 그 지령을 처리한다. 보조쉘을 작성하기전에 괄호밖에 있는 단어들이 처리된다. 즉 badstuff파일이 표준출력과 오유를 위하여 열린다. 보조쉘이 시작될 때 그것은 어미쉘로부터 표준입력, 표준출력, 표준오유를 계승한다. 이때 보조쉘은 >연산자를 처리한다. 표준출력이 파일 goodstuff로 치환된다. 출력은 goodstuff로 나가고 오유는 badstuff로 나가게 된다(그림 9-3).

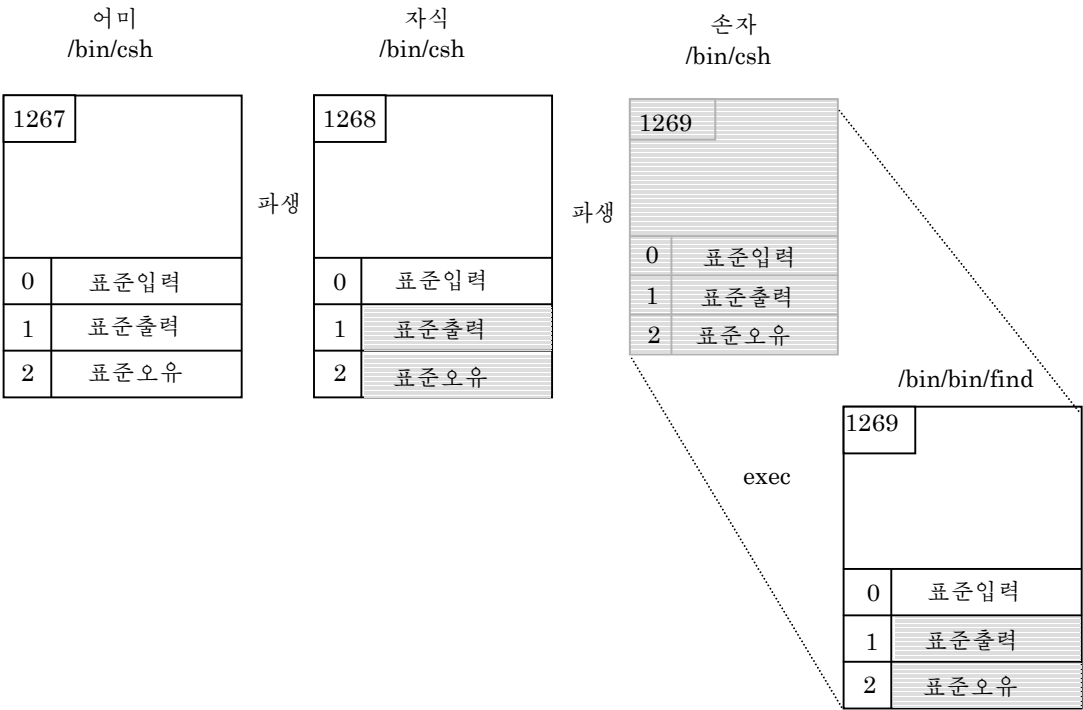



그림 9-3. 표준출력과 표준오유 분리

noclobber변수 C쉘의 특수내부변수 noclobber가 설정되면 방향바꾸기할 때 파일이 파괴되지 않게 할수 있다.

표 9-5. noclobber변수

noclobber가 설정되지 않았다.	**파일이 존재한다.**	**파일이 존재하지 않는다.**
command>file	파일이 겹쳐 진다.	파일이 작성된다.
command>>file	파일이 추가된다.	파일이 작성된다.
noclobber가 설정된다.		
command>file	오유통보	파일이 작성된다.
command>>file	파일이 추가된다.	오유통보
겹쳐쓰기 noclobber		
command>!file	noclobber가 설정되면 이 지령에 대한 그의 영향을 무시하고 파일을 열거나 자르고 지령의 출력을 파일로 방향바꾸기한다.	

(표계속)

겹쳐쓰기 noclobber	
command>>!file	noclobber변수를 무시한다. 파일이 존재하지 않으면 그것이 작성되고 지령으로부터 출력이 그에 추가된다(실례 9-44를 참고).

실례 9-44

1. % **cat filex**
   abc
   123
2. % **date > filex**
3. % **cat filex**
   Wed Aug 5 11:51:04 PDT 2001
4. % **set noclobber**
5. % **date > filex**
   filex: File exists.
6. % **ls >! filex** *# Override noclobber for this command only*
   % cat filex
   abc
   abl
   dir
   filex
   plan. c
7. % **ls > filex**
   filex: File exists.
8. % **unset noclobber** *# Turn off noclobber permanently*

설명

1. filex의 내용들이 현시장치에 현시된다.
2. date지령의 출력이 filex에 방향바꾸기된다. 파일의 끝이 잘라 지고 원래 있던 내용우에 덧씌여 진다.
3. filex의 내용들이 현시된다.
4. noclobber변수가 설정된다.
5. filex가 이미 존재하고 noclobber가 설정되므로 쉘은 파일이 존재하며 그우에 덧쓰는것을 허락하지 않는다는것을 통보한다.
6. >!연산자가 noclobber의 영향을 무시하기때문에 ls의 출력이 filex에 방향바꾸기된다.
7. >!기호의 영향은 일시적이다. 그것은 noclobber를 해제하지 못한다. 다만 그것이 사용되는 지령에 대해서만 noclobber를 무시한다.
8. noclobber변수가 해제된다.

10. 변수

C쉘변수들은 문자렬들이나 문자렬모임만을 가진다. 일부 변수들은 쉘에 내장되여 있고 noclobber나 filec변수와 같이 절환하여 설정할수 있다. path변수와 같은 다른 변수들에는 문자렬값이 대입된다. 사용자는 자체변수들을 작성하여 문자렬이나 지령들의 출력에 대답할수 있다. 변수이름들은 경우에 맞게 지우며 수자, 문자, 밑선으로 된 최고 20개 문자들까지 포함할수 있다. 2가지 형태의 변수들 즉 국부변수와 환경변수가 있다. 변수의 유효범위는 명백하다. 국부변수는 그것이 정의된 쉘에 보여 진다. 환경변수의 유효범위는 보통 전역적이다. 유효범위는 이 쉘과 그로부터 파생된 모든 프로쎄스들이다.

화페기호($)는 변수이름앞에 올 때 쉘에 그 변수의 값을 추출하라고 알려 주는 특수메타문자이다. echo지령은 인수로서 변수가 주어 질 때 쉘이 지령행을 처리하고 변수바꿔넣기를 진행한 다음에 그 변수의 값을 현시한다.

특수표기 $?는 변수이름앞에 올 때 변수가 설정되였는가를 알수 있게 한다.

만일 1이 돌려 지면 그것은 참을 의미한다. 즉 변수가 설정되였다. 령이 복귀되면 그것은 거짓을 의미한다. 즉 변수는 설정되지 않았다.

실례 9-45

```
1.  % set filec
2.  % set history = 50
3.  % set name = George
4.  % set machine = `uname –n`
5.  % echo $?machine
    1
6.  % echo     $blah
    0
```

설명

1. 파일이름완성을 위해 내부변수 filec를 설정한다.
2. 현시되는 사건들의 수를 조종하기 위해 내부변수 history를 설정한다.
3. 사용자정의된 변수 name을 George로 설정한다.
4. 사용자정의된 변수 machine을 UNIX지령의 출력으로 설정한다. 이 지령은 거꿀인용부호안에 있다. 따라서 쉘에 지령바꿔넣기를 하라고 알려 준다.
5. 변수가 설정되였는가 않았는가를 검사하기 위하여 $?를 변수이름앞에 붙인다. 검사결과가 1(참)이면 변수는 설정되였다.
6. $?의 결과는 령(거짓)이다. 변수가 설정되지 않았다.

대괄호 대괄호는 그뒤에 오는 임의의 문자들로부터 변수를 분리시킨다.

실례 9-46

```
1.  % set var = net
    % echo $var
    net
2.  % echo $varwork
```

varwork: Undefined variable.
3. % **echo ${var}work**
network

설명

1. 변수이름을 둘러 싼 대괄호는 그뒤에 오는 문자들로부터 변수를 분리시킨다.
2. varwork라고 하는 변수는 정의되지 않았다. 쉘은 오유통보를 인쇄한다.
3. 대괄호는 변수를 그에 첨가된 문자들로부터 보호한다. $var는 확장되고 문자렬 work는 추가된다.

국부변수 국부변수들은 그것들이 작성되는 쉘에서만 알려 진다. 만일 국부변수가 .cshrc파일에서 설정된다면 변수는 새로운 C쉘이 시작될 때마다 재설정된다. 습관상 국부변수들의 이름을 소문자들로 짓는다.

국부변수설정 대입되는 문자렬에 한개이상의 단어가 있다면 인용부호속에 넣어야 한다. 그렇지 않으면 첫번째 단어만이 변수에 대입된다. 갈기부호주위에 공백이 있는것은 문제로 되지 않지만 갈기부호의 한쪽에 공백이 있으면 다른쪽에도 반드시 공백이 있어야 한다.

실례 9-47

1. % **set round = world**
2. % **set name = “Santa Claus”**
3. % **echo $round**
world
4. % **echo $name**
Santa Claus
5. % **csh** *# Start a subshell*
6. % **echo $name**
name: Undefined variable.

설명

1. 국부변수 round에 값 world가 대입된다.
2. 국부변수 name에 값 santa claus가 대입된다. 2중인용부호는 쉘이 santa와 claus사이에 있는 공백을 무시하게 한다.
3. 변수앞에 붙은 $는 쉘이 변수바꿔넣기를 진행하게 한다. 즉 변수에 기억된 값을 꺼내도록 한다.
4. 변수바꿔넣기가 진행된다.
5. 새로운 C쉘(보조쉘이라고 한다.)프로세스가 시작된다.
6. 보조쉘에서 변수 name은 정의되지 않았다. 그것은 국부변수로 어미쉘에서 정의되였다.

set지령 set지령은 쉘을 위해 설정된 모든 국부변수들을 인쇄한다.

실례 9-48

(The Command Line)
% **set**
argv ()
cwd /home/jody/ellie
fignore .o
filec
history 500
home /home/jody/ellie
hostname jody
ignoreef
noclobber
notify
path (/home/jody/ellie /bin /usr/local /usr/usr/bin/usr/etc .)
prompt jody%
shell /bin/csh
status 0
term sun-cmd
user ellie

설명

이 쉘을 위해 설정된 모든 국부변수들이 인쇄된다. 변수들의 대부분이 .csherc 파일에서 설정된다. argv, shell, term, user와 status변수들은 미리 설정된 변수 즉 내부변수이다.

내부국부변수 쉘은 많은 미리 정의된 변수들을 가지고 있다. 일부 변수들은 절환된다. 실례로 noclobber를 설정한다면 그 변수는 사용할수 있다.

그러나 noclobber의 설정을 해제하면 그것을 사용할수 없다. 일부 변수들은 설정될 때 정의를 요구한다. 내부변수들을 다른 C쉘에서도 사용하려고 한다면 그것들을 보통 .cshrc파일에서 설정한다. 이미 취급된 내부변수들로는 noclobber, cdpath, history .filec와 noglob가 있다. 표 9-16에 그에 대한 완전한 목록을 보여 주었다.

환경변수 환경변수들은 흔히 전역변수라고 한다. 그것들은 작성되는 쉘에서 정의되고 그 쉘로부터 파생되는 모든 쉘들에 의해 계승된다.

환경변수들은 보조쉘들에 의해서 계승되지만 보조쉘에서 정의된 변수들은 어미쉘로 넘겨 지지 않는다. 계승은 어미로부터 자식에로 이루어 질수 있으나 반대로는 되지 않는다(실생활과 같이). 습관상 환경변수들은 대문자로 이름 짓는다.

실례 9-49

(The Command Line)
1. % **setenv TERM wyse**
2. % **setenv PERSON "Joe Jr."**

3. % **echo $TERM**
   wyse
4. % **echo $PERSON**
   Joe Jr.
5. % **echo $$** *# $$ evaluates to the PID of the current shell*
   206
6. % **csh** *# Start a subshell*
7. % **echo $$**
   211
8. % **echo $PERSON**
   Joe Jr.
9. % **setenv PERSON "Nelly Nerd"**
10. % **echo $PERSON**
    % Nelly Nerd
11. % **exit** *# Exit the subshell*
12. % **echo $$**
    206
13. % echo $PERSON *# Back in parent shell*
    Joe Jr.

설명

1. 쉘환경변수 TERM이 wyse말단으로 설정된다.
2. 사용자정의된 변수 PERSON은 Joe Jr로 설정된다. 인용부호를 사용하여 공백을 무시한다.
3. 변수이름앞에 붙은 표식 $는 쉘이 변수의 내용들을 평가하게 하는데 이것을 변수바꿔넣기라고 한다.
4. 환경변수 PERSON의 값이 인쇄된다.
5. $$변수는 현재쉘의 PID를 가지고 있다. PID는 206이다.
6. csh지령은 보조쉘인 새로운 쉘을 파생한다.
7. 현재쉘의 PID가 인쇄된다. 이것은 새로운 C쉘이므로 다른 PID번호를 가진다. PID는 211이다.
8. 환경변수 PERSON이 새로운 쉘에 의해 계승되였다.
9. PERSON변수가 Nelly Nerd로 재설정된다. 이 변수는 쉘로부터 파생된 임의의 쉘들에 의해 계승된다.
10. PERSON변수의 새로운 값이 인쇄된다.
11. 이 C쉘이 탈퇴된다.
12. 초기 C쉘이 실행되고 있다. PID206이 인쇄되여 그것을 보여 준다. 그것은 보조쉘이 파생되기전의 값과 같다.
13. PERSON변수가 자기의 초기값을 가진다.

환경변수의 인쇄 printenv(UCB)와 env(SVR4)지령들은 쉘과 그의 보조쉘들을 위해 설정된 모든 환경변수들을 인쇄한다. setenv지령은 C쉘의 UCB와 SVR4판본에서 변수들과 그의 값들을 인쇄한다.

실례 9-50

```
% env
FONTPATH=/usr/local/OW3/lib/fonts
HELPPATH=/usr/local/OW3/lib/locale:/usr/local/OW3/llb/help
HOME=/home/jody/ellie
LD_LIBRARY_PATH=/usr/local/OW3/lib
LOGNAME=ellie
MANPATH=/ur/local/man:/usr/local/man:/usr/local/doctools/man:/usr/man
NOSUNVIEW=0
OPBNWINHOME=/usr/local/OW3
PATH=/bin:/usr/local:/usr:/usr/bin:/usr/etc:/home/5bin:/usr/
    doctools:/usr:.
PWD=/home/jody/ellie
SHELL=/bin/csh
TERM=sun-cmd
USER=ellie
WINDOW_PARENT=/dev/win0
WINDOW_TTYPARMS=
WMGR_ENV_PLACEHOLDER=/dev/win3
```

설명

환경변수들은 이 대화조종과 쉘로부터 파생되는 모든 프로쎄스들을 위해 설정된다. 응용프로그람들은 환경변수들의 설정을 요구한다. 실례로 man지령은 사용지도서페지가 있는 위치에 설정된 MANPATH변수를 가지며 openwin프로그람은 그 폰트들이 기억되여 있는 위치에 설정된 환경변수를 가진다. 이 프로그람들이 실행될 때 변수들에 있는 정보가 그것들에게 넘겨 진다.

배렬 C쉘에서 배렬은 공백이나 타브로 분리되고 괄호로 닫기는 단어들의 목록이다. 보조스크립트들에 의해서 1부터 시작하여 배렬의 요소들에 번호가 매겨 진다.

만일 보조스크립트에 대한 배렬요소가 하나도 없다면 통보 Subscript out of range가 현시된다. 지령바꿔넣기도 배렬을 작성한다. 만일 $#표기가 배렬이름앞에 붙으면 배렬에 있는 요소들의 수가 현시된다.

실례 9-51

```
1. % set fruit = ( apples pears peaches plums )
2. % echo $fruit
   apples pears peaches plums
3. % echo $fruit[l]          # Subscripts start at 1
   appies
4. % echo $£i-uit[2-4]       # Prints the 2nd, 3rd, and 4th elements pears
   peaches plums
5. $ echo $fruit[6]
   Subscript out of range.
```

6. % **echo $fruit[*]** *# Prints all elements of. the array apples pears peaches plums*
7. % **echo $#fniit** *# Prints the number* of *elements*
   4
8. % **echo $£ruit[$#fruit]** *# Prints the last element*
   plums
9. % **set fruit[2] = bananas** *# Reassigns the second element*
   % **echo $£ruit**
   apples bananas peaches plums
10. % **set path = (- /usr/bin /usr /usr/local/bin .)**
    % **echo $path**
    /home/Jody/ellie /usr/bin /usr /usr/local/bin
11. % **echo $path[1]**
    /home/jody/el1ie

설명

1. 단어목록이 괄호로 닫겨 진다. 매 단어는 공백으로 분리된다. 이 배렬은 fruit라고 한다.
2. fruit배렬에 있는 단어들이 인쇄된다.
3. fruit배렬의 첫번째 요소가 인쇄된다. 보조스크립트들은 1에서 시작된다.
4. 단어목록의 두번째, 세번째, 네번째 요소들이 인쇄된다. 횡선으로 범위를 지정한다.
5. 배렬은 6개의 요소들을 가지지 않는다. 보조스크립트가 범위에서 벗어난다.
6. fruit배렬의 모든 요소들이 인쇄된다.
7. 배렬앞에 붙은 $#표식을 사용하여 배렬에 있는 요소들의 수를 얻는다. fruit배렬에는 4개의 요소들이 있다.
8. 보조스크립트 $#fruit가 배렬에 있는 요소들의 총수와 같으므로 그 값이 배렬 즉 [$#fruit]의 첨수값으로 사용되면 fruit배렬의 마지막요소가 인쇄된다.
9. 배렬의 두번째 요소가 새로운 값으로 대입된다. 배렬이 치환된 값 bananas 와 함께 인쇄된다.
10. path변수는 지령들을 탐색하는데 사용되는 등록부들로 이루어 진 C쉘의 특수한 배렬이다. 배렬을 작성하여 경로의 개개의 요소들이 호출되거나 변화될수 있다.
11. path의 첫번째 요소가 인쇄된다.

shift지령과 배렬 내부지령 shift는 배렬이름을 인수로서 가지면 배렬의 첫번째 요소를 자리옮김한다. 배렬의 길이가 하나씩 감소된다(인수가 없는 shift지령은 내부 argv배렬의 첫번째 요소를 자리옮김한다. 362페지의 《지령행인수》를 보여 주었다.).

실례 9-52

1. **% set names = (Mark Tom Liz Dan Jody)**
2. **% echo $names**
   Mark Tom Liz Jody
3. **% echo $names[1]**
   Mark
4. **% shift names**
5. **% echo $names**
   Tom Liz Dan Jody
6. **% echo $names[1]**
   Tom
7. **% set days = (Monday Tuesday)**
8. **% shift days**
9. **% echo $days**
   Tuesday
10. **% shift days**
11. **% echo $days**
12. **% shift days**
    shifts: no more words.

설명

1. 배렬은 names라고 한다. 그것에 괄호안에 있는 단어들의 목록이 대입된다. 매 단어는 공백으로 분리된다.
2. 배렬이 인쇄된다.
3. 배렬의 첫번째 요소가 인쇄된다.
4. 배렬은 한 요소만을 왼쪽으로 자리옮김한다. 단어 Mark가 자리옮김된다.
5. 배렬은 자리옮김후에 한개 요소가 감소되였다.
6. 자리옮김후에 배렬의 첫번째 요소는 Tom이다.
7. days라고 하는 배렬이 작성된다. 그것은 2개의 요소들 Monday와 Tuesday를 가진다.
8. 배렬 days는 왼쪽으로 하나 자리옮김된다.
9. 배렬이 인쇄된다. 그 배렬에 Tuesday만 남는다.
10. 배렬 days는 다시 자리옮김된다. 배렬은 비게 된다.
11. days배렬이 빈다.
12. 이때 자리옮김을 하면 쉘은 빈 배렬로부터 원소들을 자리옮김할수 없다는 것을 현시하는 오유통보를 보낸다.

문자렬로부터 배렬만들기 인용부호안에 있는 문자렬로부터 단어목록을 작성하려고

할수 있다. 이때 괄호안에 문자렬변수를 넣어 그것을 실현할수 있다.

실례 9-53

1. **% set name = “Thomas Ben Savage”**
   % echo $name[1]
   Thomas Ben Savage
2. **% echo $name[2]**
   Subscript out of range.
3. **% set name = ($name)**
4. **% echo $name[l] $name[2] $name[3]**
   Thomas Ben Savage

설명

1. 변수 name에 문자렬 Thomas ben Savage를 대입한다.
2. 배렬로 취급될 때 한개의 요소인 전체 문자렬만이 있다.
3. 변수를 괄호안에 넣어 name이라고 하는 단어들의 배렬을 작성한다.
4. 새로운 배렬의 3개 요소들이 현시된다.

11. 특수변수

한개문자로 이루어 진 여러개의 변수들이 C쉘에 내장되여 있다. 문자앞에 붙은 $는 변수를 해석한다(표 9-6).

표 9-6. 변수들과 그 의미

변 수	실 례	의 미
$?var	echo$?name	변수가 설정되였으면 1을, 아니면 0을 되돌린다.
$#var	echo$#fruit	배렬에 있는 요소들의 수를 인쇄한다.
$$	echo$$	현재쉘의 PID를 인쇄한다.
$<	setname=$<	사용자로부터 행바꾸기건을 누를 때마다 입력행을 받는다.

실례 9-54

1. **% set num**
   % echo $?num
   1
2. **% echo $path**
   /home/jody/ellie   /usr/bin   usr/local/bin
   % echo $#path
   3

3. % **echo $$**
245

% **csh** *# Start a subshell*
% **echo $$**
248

4. % **set name = $<**
Christy Campbell
% **echo $name**
Chisty Campbell

설명

1. 변수 num이 빈값으로 설정된다. $?가 앞에 붙는 변수는 설정되였으면(빈값이나 어떤 값으로) 1로 되고 설정되지 않았다면 0으로 된다.
2. path변수가 인쇄된다. 그것은 3개의 요소들을 가진 배렬이다. $#앞에 붙은 변수는 배렬에 있는 요소들의 수를 추출하여 인쇄한다.
3. $$는 현재프로쎄스인데 이 경우 C쉘의 PID이다.
4. $<변수는 사용자로부터 입력행을 받지만 행바꾸기를 포함하지 않고 그 행을 변수 name에 기억시킨다. name변수의 값이 현시된다.

경로이름변수변경자 경로이름이 변수에 대입되면 특수C쉘확장자를 그에 추가하여 경로이름변수를 조작할수 있다. 경로이름은 4개의 부분 즉 머리, 꼬리, 뿌리와 확장자로 나누어 진다. 경로이름변경자들의 실례와 그 기능을 표 9-7에 보여 주었다.

set pn=/home/ellie/prog/check. c

표 9-7. 경로이름변경자

변경자	의 미	실 례	결 과
:r	뿌리	echo$pn:r	/home/ellie/prog/check
:h	머리	echo$pn:h	/home/ellie/prog
:t	꼬리	echo$pn:t	check.c
:e	확장자	echo$pon:e	c
:f	대역	echo$p:gt	(실례 9-55를 참고)

실례 9-55

1. % **set pathvar = /home/danny/program.c**

2. % **echo $pathvar;r**
/home/danny/program

3. % **echo $pathvar:h**
/home /danny

4. % **echo $pathvar:t**
   program.c
5. % **echo $pathvar:e**
   c
6. % **set pathvar = (/home/*)**
   echo $pathvar
   /home/Jody /home/local /home/lost+found /home/peri /home/imp
7. % **echo $pathvar:gt**
   Jody local lost+found peri tmp

설명

1. 변수 pathvar가 /home/danny/program.c로 설정된다.
2. :r가 변수에 붙으면 현시될 때 확장자가 지우기된다.
3. :h가 변수에 붙으면 경로의 머리가 현시된다. 즉 경로의 마지막요소가 지어진다.
4. :t가 변수에 붙으면 경로의 꼬리끝(마지막요소)이 현시된다.
5. :e가 변수에 붙으면 확장자가 현시된다.
6. 변수가 /home/*로 설정된다. 별표는 /home/으로 시작하는 현재등록부에 있는 모든 경로이름들로 확장된다.
7. :gt가 붙으면 매(대역)경로요소들의 꼬리끝이 현시된다.

12. 지령바꿔넣기

지령을 거꿀인용부호안에 넣어 문자렬이나 변수에 UNIX지령의 출력을 대입할수 있다. 이것을 지령바꿔넣기라고 한다(건반에서 거꿀인용부호는 보통 물결표기호아래에 있다.). 만일 지령의 출력이 변수에 대입되면 그것은 문자렬이 아니라 단어목록(341페지의 《배렬》을 참고)으로 기억되여 목록에 있는 매 단어들이 개별적으로 호출될수 있다. 목록으로부터 단어를 호출하기 위해 보조스크립트를 변수이름에 붙인다.

보조스크립트는 1에서 시작한다.

실례 9-56

1. **%** echo The name of my machine is 'uname -n' .
   The name of my machine is stardust.
2. % **echo The present working directory is 'pwd'.**
   The present working directory is /home/stardust/john.
3. **%** set **d = 'date'**
   % **echo $d**
   Sat Jun 20 14:24:21 PDT 2001
4. **%** echo $d[2] $d[6]
   Jun 2001

5. **% set d = "'date'"**
   % echo $d[1]
   Jun 20 14:24:21 PDT 2001

설명

1. UNIX지령 uname-n이 거꿀인용부호로 닫겨 있다. 쉘은 거꿀인용부호를 발견하면 그안에 있는 지령 uname-n을 실행시키고 지령의 출력 stardust를 문자렬에로 치환한다. echo지령이 그 인수들을 표준출력으로 인쇄할 때 콤퓨터의 이름은 그 인수들중 어느 하나가 될수 있다.
2. UNIX지령 pwd가 쉘에 의해서 실행되고 출력이 문자렬안의 그 위치에 치환된다.
3. 국부변수 d에 dete지령의 출력이 대입된다. 출력이 단어들의 목록(배렬)으로 기억된다.
4. d배렬의 요소 2와 6이 인쇄된다. 보조스크립트가 1에서 시작된다.
5. 출력이 2중인용부호속에 있으므로 단어목록이 아니라 한개의 문자렬로 된다.

단어목록들과 지령바꿔넣기 지령이 거꿀인용부호안에 있고 변수에 대입될 때 그 결과값은 배렬(단어목록)이다.

배렬의 매 요소는 배렬이름에 보조스크립트를 붙여 호출할수 있다. 보조스크립트는 1에서 시작한다. 배렬에 있는 단어들의 수보다 더 많은 보조스크립트가 사용되면 C쉘은 Subscript out of range를 인쇄한다. 만일 지령의 출력이 한개이상의 행으로 이루어 진다면 행바꾸기들은 매행으로부터 없어 지고 한개의 공백으로 교체된다.

실례 9-57

1. **% set d = 'date'**
   % echo $d

   Fri Aug 29 14:04:49 PDT 20013
2. **% echo $d[1-3]**
   Fri Aug 29
3. **% echo $d[6]**
   2001
4. **% echo $d[7]**
   Subscript out of range.
5. **% echo The calendar for the month of November le 'cal 11 2001"**
   The calendar for month of November is November 2001 S M Tu W Th F S 1 2 3 4 5 6 7 8. 9 10 11 12 13 14 15 16 17 18. 19 20 21 22 23 24 25 26 27 28. 29 30

설명

1. 변수 d에 UNIXdate지령의 출력이 대입된다. 그 출력은 배렬로서 기억된

다. 변수의 값이 현시된다.
2. 배렬의 첫번째 3개 요소들이 현시된다.
3. 배렬의 6번째 요소가 현시된다.
4. 배렬에는 7번째 요소가 없다. 쉘은 보조쉘이 범위를 벗어 난다고 통보한다.
5. 출력은 한개이상의 행으로 현시된다. 매 행바꾸기가 한개의 공백으로 교체된다. 이것은 기대했던 출력이 아닐수 있다.

실례 9-58

```
1.  % set   machine = `rusers | awk '/tom/{print $1}

2.  % echo machine
    dumbo bambi dolphin

3.  % echo $ttmachine
    3

4.  % echo $inachine[$#machine]
    dolphin

5.  % echo $machine
    dumbo bambi dolphin

6.  % shift $machine
    % echo $machine
    bambi dolphin

7.  % echo $machine[1]
    bambi

8.  % echo $#machine
    2
```

설명

1. rusers지령의 출력은 awk로 파이프된다. 만일 정규식 tom이 발견되면 awk는 첫번째 마당을 인쇄한다. 이 경우 첫번째 마당은 사용자 tom이 등록가입된 콤퓨터이름이다.
2. 사용자 tom이 3개의 콤퓨터에 등록가입된다. 콤퓨터이름들이 현시된다.
3. 배렬에 있는 요소들의 수가 이름앞에 $#를 붙여 호출된다. 배렬에 3개의 요소가 있다.
4. 배렬의 마지막요소가 현시된다. 배렬에 있는 요소들의 수($#machine)가 보조스크립트로 사용된다.
5. 배렬이 현시된다.
6. shift지령이 배렬을 왼쪽으로 자리옮김한다. 배렬의 첫번째 요소가 밀려 나고 보조스크립트들은 1부터 시작하여 번호를 매긴다.
7. shift다음에 배렬의 첫번째 요소가 현시된다.

8. shift다음에 배렬의 길이가 한개 감소되였다.

13. 인용부호찍기

C쉘은 특수한 의미를 가지는 메타문자들의 모임을 가지고 있다. 사실 자모문자나 수자가 아닌 건반에 있는 거의 모든 임의의 문자는 쉘에 대해서 특수한 의미를 가진다. 아래에 그것들의 부분목록을 현시한다.

* ? [ ] $ ~ ! ^ & { } ( ) > < / ; : %

거꿀빗선과 인용부호를 사용하여 쉘이 메타문자들을 해석하지 않게 한다. 거꿀빗선은 한개 문자를, 인용부호는 문자들의 렬을 해석되지 않게 보호한다. 인용부호사용에 관한 일반규칙은 다음과 같다.

1. 인용부호는 쌍으로 되고 한개 행에서 일치되여야 한다. 거꿀빗선을 사용하여 새행을 벗어 나면 인용부호가 다음행에서 일치될수 있다.
2. 단일인용부호는 2중괄호를 보호하고 2중인용부호는 단일인용부호를 보호한다.
3. 단일인용부호는 리력문자(!)를 제외하고 모든 메타문자들을 해석되지 않게 한다.
4. 2중인용부호는 리력문자(!), 변수바꿔넣기문자($) 그리고 거꿀인용부호(지령바꿔넣기를 위해 사용된다.)를 제외한 모든 메타문자들이 해석되지 않게 한다.

거꿀빗선 거꿀빗선을 사용하여 한개문자가 해석되지 않도록 하는데 C쉘에서는 거꿀빗선이 리력문자인 감탄부호를 탈퇴시키는데 사용되는 유일한 문자이다. 거꿀빗선은 흔히 행바꾸기를 탈퇴시키는데 사용된다. 거꿀빗선은 인용부호안에서 해석되지 않는다.

실례 9-59

1. % **echo Who are you?**
   echo: No match.
2. % **echo Who are you\ ?**
   Who are you?
3. % **echo This is a very,very long line and this is where I\ break the line.**
   This is a very, very long line and this is where I
   I break the line.
4. % **echo "\ \ abc"**
   \ \ *abc*
   % **echo '\ \ abc'-**
   \ \ *abc*
   % **echo \ \ abc**
   \ *abc*

설명

1. 물음표가 파일이름확장에 사용된다. 그것은 한개 문자를 대신한다. 쉘은 현재등록부에서 you로 시작되고 그다음에 한개 문자가 더 오는 파일을 찾는

다. 등록부에 그러한 이름의 파일이 없으므로 쉘은 No match를 인쇄하여 그것과 대응되는 파일이 없다는것을 통보한다.
2. 쉘은 물음표가 거꿀빗선에 의해서 메타문자로 되지 않으므로 그것을 해석하려고 하지 않는다.
3. 거꿀빗선에 의하여 문자렬이 다음행으로 계속된다.
4. 거꿀빗선이 단일인용부호나 2중인용부호속에 있으면 그것이 인쇄된다. 인용부호속에 있지 않으면 거꿀빗선은 거꿀빗선을 벗어 나게 한다.

단일인용부호 단일인용부호는 같은 행에서 일치되여야 하고 리력문자(!)를 제외한 모든 메타문자들이 문자그대로 사용되게 한다. C쉘은 거꿀빗선을 해석하기전이 아니라 인용부호를 해석하기전에 리력문자를 해석하기때문에 리력문자는 메타문자로 쓰인다.

실례 9-60

1. % **echo 'I need $5.00'**
   I need $5.00
2. % **echo ' I need $500.00 now\ ! \ ! '**
   I need $500.00 now!!!
3. % **echo 'This is going to be a long line so**
   Unmatched '.
4. % **echo 'This is going to be a long line so **
   I used the backslash to suppress the newline'
   This is going to a long line so
   I used the backslash to suppress the newline

설명

1. 문자렬은 단일인용부호로 닫긴다. 리력문자를 제외하고 모든 문자들은 쉘해석으로부터 보호된다.
2. !!는 거꿀빗선을 사용하여 쉘해석으로부터 보호된다.
3. 인용부호는 같은 행에서 일치되여야 한다. 그렇지 않으면 쉘은 unmatched를 통보한다. 만일 행이 계속된다면 거꿀빗선을 사용하여 다음행으로 계속되게 한다. 인용부호는 다음행의 끝에서 닫긴다. 쉘은 행바꾸기를 무시했지만 echo지령은 무시하지 않는다.

2중인용부호 2중인용부호는 일치되여야 하고 변수와 지령바꿔넣기를 하게 하며 리력문자(!)를 제외한 모든것을 해석으로부터 보호한다. 거꿀빗선은 2중인용부호속에 있을 때 화페기호 $를 메타문자로 사용할수 없게 하지는 못한다.

실례 9-61

1. % **set name = Bob**
   % **echo "Hi $name"**
   Hi Bob
2. % **echo "I don't have time."**

I don't have time.

3. % **echo "WOW!"** *# Watch the history metacharacter!*
   ": Event not found.

4. % **echo "Whoopie\ !"**
   Whoopie!

5. % **echo "I need \ $5.00"**
   I need \ .00

설명

1. 국부변수 name에 값 bob를 대입한다. 2중인용부호는 화페기호가 변수교체에 사용되게 한다.
2. 단일인용부호는 2중인용부호안에서 보호된다.
3. 2중 혹은 단일인용부호는 쉘해석으로부터 감탄부호를 보호하지 못한다. history내부지령은 2중인용부호로 시작하는 마지막지령을 찾는데 그 사건은 발견되지 않았다.
4. 거꿀빗선은 감탄부호를 보호하는데 사용한다.
5. 거꿀빗선은 화페기호가 2중인용부호안에서 사용될 때 그것은 특수한 의미에서 벗어 나지 못한다.

인용부호찍기유희 인용부호찍기규칙을 지키면 2중인용부호와 단일인용부호는 단일지령에서 여러가지 조합들로 사용될수 있다.

실례 9-62

1. % set **name = Tom**

2. % **echo "I can't give $name" ' $5.00\ !'**
   I can't give Tom $5.00!

3. % **echo She cried, \ "0h help me\ !' "', $name.**
   She cried, "Oh help me!", Tom.

설명

1. 국부변수name에 Tom이 대입된다.
2. 단어 can't에 있는 단일인용부호는 2중인용부호속에 있을 때 보호된다. 만일 $5.00에 있는 화페기호 $가 2중인용부호속에 있으면 쉘은 변수바꿔넣기를 진행하려 한다. 그러나 $5.00은 단일인용부호속에 있으므로 $는 문자그대로 사용된다. 2중인용부호와 단일인용부호는 감탄부호를 쉘해석으로부터 보호할수 없으므로 거꿀빗선을 사용하여 감탄부호를 보호한다.
3. 첫번째 인용부호가 거꿀빗선에 의해 보호된다. 감탄부호도 거꿀빗선에 의해 보호된다. 마지막인용부호는 단일인용부호속에 있다. 단일인용부호는 2중인용부호를 보호한다.

인용부호를 옳게 사용하기 위한 단계 더 복잡한 지령에서는 보통 여기서 설명하는

단계들을 따르지 않으면 인용부호들을 정확히 사용할수 없다(부록 3을 참고).

1. UNIX지령과 그 문법들을 학습해야 한다. 변수바꿔넣기전에 기대하는 결과를 얻는가 보기 위해 값들을 지령행에로 넘긴다.

   %namk-f:'/^zippy pinhead/{print"phone is"$2}'datafile 408.12}-4563

2. 만일 UNIX지령이 정확히 동작했다면 변수들을 넣는다. 이때 인용부호들을 지우거나 변화시키지 말아야 한다. 변수들을 그것들이 표시하는 단어대신에 넣는다. 이 실례에서 zippy pinhead를 $name으로 교체한다.

   % set name="zippy pinhead"
   % nawk-F: '/^$name/{print "phone is"$2}'datafile

3. 인용부호찍기유희를 다음과 같이 한다.
   첫번째 단일인용부호를 왼쪽에서 시작해서 $name에 있는 표식 $앞에 닫는 단일인용부호를 삽입한다. 그렇게 되면 인용부호가 일치된다.

   nawk -F:'/^'$name/{print"phone is "$2}' datafile

   다음에 $name의 마지막문자 e의 오른쪽에 다른 단일인용부호를 넣는다. 이 단일인용부호는 닫는 대괄호뒤에 있는 단일인용부호와 대응된다.

   %nawk-F:'/^'$name'/{print "phone is "$2}'datafile

   왼쪽에서부터 시작하여 단일인용부호의 수를 계수한다. 그 수는 우수로서 4이다. 매개 쌍의 단일인용부호안에 있는 모든것이 쉘에 의해 무시된다. 인용부호들은 다음과 같이 일치된다.

   %nawk-F:'$1~/'$name'/{print $2}' filename

4. 마지막단계: 변수들을 2중인용부호에 넣는다. 매 변수를 한쌍의 2중인용부호속에 편리하게 넣는다. 2중인용부호는 확장된 변수에 있는 공백을 보호한다. 실례로 zippy pingead에 있는 공백이 보호된다.

   %nawk-F:'s1~/'"$name"'/{print $2}'filename

변수들을 인용부호안에 넣기 변수들을 인용부호안에 넣는것이 필요할 때 :x와 :q 변경자를 사용한다.

:q변경자로 인용부호찍기 :q변경자를 사용하여 2중인용부호를 대신한다.

실례 9-63

```
1. % set name = "Daniel Savage"
2. % grep $name:q database
   same as
3. % grep "$name" database
4. % set food = "apple pie"
5. % set dessert = ( $food "ice cream")
```

6. % **echo $#dessert**
   3
7. % **echo $dessert[l]**
   apple
8. % **echo $dessert[2]**
   pie
9. % **echo $dessert[3]**
   ice cream
10. % **set dessert = ($food8.g "ice cream")**
11. % **echo $#dessert**
   2
12. % **echo $dessert[l]**
   apple pie
13. % **echo $dessert[2]**
   ice cream

설명

1. 변수에 문자렬 Daniel Savage가 대입된다.
2. ：q가 변수에 추가될 때 변수는 인용부호속에 넣어 진다. 이것은 변수를 2중인용부호속에 넣는것과 갈다.
3. 변수 $name을 둘러 싼 2중인용부호는 변수가 대입되게 하지만 임의의 공백문자를 보호한다. 2중인용부호가 없으면 grep프로그람은 Savage라고 하는 파일과 database라고 하는 파일에서 Daniel을 탐색한다.
4. 변수 food에 문자렬 apple pie가 대입된다.
5. 변수 desert에 appile pie와 ice cream으로 이루어 진 배렬 [단어목록]이 대입된다.
6. desert배렬의 요소수는 3이다. food변수가 확장되였을 때 인용부호는 지우기된다. 3개의 요소 apple, pie, ice cream이 있다.
7. 배렬의 첫번째 요소가 인쇄된다. 만일 변수가 인용부호안에 닫겨 있지 않으면 분리된 단어들로 확장된다.
8. 배렬의 두번째 요소가 인쇄된다.
9. "ice cream"이 인용부호안에 있으므로 한개 단어로 취급된다.
10. desert배렬에 apple pic와 ice cream이 대입된다. :q를 사용하여 2중인용부호와 같이 변수를 인용부호안에 넣을수 있다. 즉 $food:q는 "$food"와 갈다.
11. 배렬은 2개의 문자렬 apple pie와 ice cream으로 이루어 진다.
12. 배렬의 첫번째 요소 apple pie가 인쇄된다.
13. 배렬의 두번째 요소 ice cream이 인쇄된다.

:x변경자로 인용부호찍기 만일 배렬을 작성하고 그 목록에 있는 어떤 단어가 메타문자들을 포함한다면 :x는 쉘이 변수바꿔넣기를 진행할 때 메타문자들을 해석하지 않게 한다.

실례 9-64

1. **% set things = "*.c** a?? **£ile[l-5]"**
   % echo $#things
   1
2. **% set newthings = ($things)**
   set; No match.
3. **% set newthings = ($tbings:x)**
4. **% echo $#newthings**
   3
5. **% echo "$newthings[l] $newthings[21 $newthings[3] "**
   **.c a?? file[l-5]*
6 **% grep $newthings[2]:<i filex**
The question marks in a?? would be used for filename expansion it is not quoted

설명

1. 변수 things에 한개 문자렬이 대입된다. 매 문자렬은 한개의 통용기호를 포함한다. 변수에 있는 요소들의 수는 한개로서 한개 문자렬이다.
2. C쉘은 문자렬 things로 배렬을 작성하려고 할 때 통용기호문자들을 확장하여 things내에서 파일이름바꿔넣기를 진행하고 No match를 내보낸다.
3. :x확장자는 쉘이 things변수에서 통용기호를 확장하지 못하게 한다.
4. 배렬 newthings는 3개의 요소들로 이루어 진다.
5. 배렬의 요소들을 인쇄하기 위해 그것들을 인용부호속에 넣어야 한다. 그렇지 않으면 쉘은 통용기호를 확장하려고 한다.
6. :q는 2중인용부호처럼 변수를 인용부호속에 넣는다. grape프로그람은 파일 filex에서 패턴 a??를 포함하는 모든 행들을 인쇄한다.

제 2 절. C쉘에 의한 프로그람작성

1. 쉘스크립트작성단계

쉘스크립트는 보통 편집기에서 작성되고 설명문들이 삽입된 지령들로 이루어 진다. 설명문앞에는 #표식이 붙는데 그것은 무엇을 하려고 하는가를 설명해 주는 본문으로 되여 있다.

첫번째 행 왼쪽의 제일 웃구석에 있는 #!이 앞에 붙은 행은 스크립트에서 행들을 실행하는 프로그람을 현시한다. 이 행은 보통 #!/bin/csh이다.

신비수라고도 하는 표식 #!는 스크립트에서 행들을 해석하는 프로그람을 핵심부가 식별하도록 하는데 사용된다. 만일 첫번째 행이 2진자료이라면 프로그람은 번역된 프로그람으로 실행된다. 만일 첫번째 행이 #!을 포함한다면 핵심부는 #!이 앞에 붙은 경로를 찾아서 그 프로그람을 해석프로그람으로 시동한다. 만일 경로가 /bin/csh라면 C쉘

이 프로그람에 있는 행들을 해석한다. 이 행은 반드시 스크립트의 제일 첫번째 행으로 되여야 한다. 그렇지 않으면 이 행은 설명문으로 취급된다.

스크립트가 시작할 때 .cshrc파일이 처음으로 읽어 지고 실행되여 파일안에 설정된 모든것이 그 스크립트의 부분으로 된다. C쉘프로그람에 대하여 -f(fast)추가선택을 사용하여 .cshrc가 스크립트로 읽어 지지 않도록 할수 있다. 이 추가선택은 다음과 같이 쓸수 있다.

```
#!/bin/csh -f
```

설명문 설명문은 표식(#)이 앞에 붙은 행들이다. 그것들은 스크립트를 설명하는데 사용된다. 때때로 스크립트에 대한 설명이 없으면 그것이 무엇을 하는지 리해하기가 힘들다.

비록 설명문이 중요하지만 그것들은 너무 빈약하거나 사용되지 않을수 있다.

자신을 위해서뿐아니라 남들을 위해서 무엇을 하고 있는가를 설명하는데 습관되여야 한다. 이제부터 아틀동안 무엇을 했는지 잘 알지 못할수 있다.

스크립트를 실행형으로 만들기 파일을 작성할 때 거기에 실행허가는 주어 지지 않는다. 스크립트를 실행시키기 위해서 허가가 필요하다. chmod지령을 사용하여 실행허가를 얻는다.

실례 9-65

1. % **chmod +x myscript**
2. % ls –lf myscript
   *-rwxr –xr 1 ellie 0 Jul 13:00 myscript**

설명

1. chmod지령을 사용하여 사용자, 그룹, 기타를 위한 실행허가를 얻는다.
2. ls지령의 출력은 모든 사용자가 myscript파일에 대한 실행허가를 가진다는 것을 나타낸다. 파일이름끝에 있는 별표(-F추가선택로부터 생기는)도 이것이 실행용프로그람이라는것을 표시한다.

스크립트작성대화조종의 실례 다음의 실례에서 사용자는 편집기에서 스크립트를 작성한다. 파일을 기억한후 chmod지령으로 실행허가를 얻어 스크립트가 실행된다. 프로그람에 오유가 있으면 C쉘이 즉시에 응답한다.

실례 9-66

```
(The Script - info)
    #!/bin/csh –f
    # This script is called info
1.  echo Hello ${LOGNAME}!
2.  echo The hour is 'date +%H’
3.  echo “This machine is 'uname -n'”
4.  echo The calendar for this “onth is
5.  cal
6.  echo The processes you are running are:
7.  ps -ef | grep  “^ *$LOGNAME' echo “Thanks for coming. See you soon\ !\ !
```

(The Command Line)

```
% chmod +x info
% info
1. Hello ellie!
2. The hour is 09
3. This machine is jody
4. The calendar for this month is
5.        July 2001
   S   M   Tu   W   Th   F   S
                     1   2   3
   4   5   6    7    8   9  10
  11  12  13   14   15  16  17
  18  19  20   21   22  23  24
  25  26  27   28   29  30  31
7. The processes you are running are:
   < output of ps prints here >
8. Thanks for coming. See you soon!!
```

설명

1. 사용자에게 인사한다. 변수 LOGNAME은 사용자이름을 가진다. BSD체계에서 USER가 사용된다. 대괄호는 변수를 감탄부호로부터 보호한다. 감탄부호에 붙은 문자가 없으면 리력문자로 해석되지 않기때문에 메타문자로 사용되지 못하게 할 필요가 없다.
2. data지령이 거꿀인용부호속에 놓인다. 쉘은 지령바꿔넣기를 진행하고 data출력인 현재시간이 출력문자렬에 교체된다.
3. uname-n지령이 콤퓨터이름을 현시한다.
4. 쉘이 지령바꿔넣기를 진행할 때 행바꾸기들은 모두 출력으로부터 지우기되므로 cal지령을 거꿀인용부호속에 넣지 않는다. 이렇게 하면 이상하게 보이는 달력이 만들어 진다. cal지령을 행에 단독으로 놓아 형식화를 보존한다.
5. 이달의 력서가 인쇄된다.
6. 사용자의 프로쎄스들이 인쇄된다. BSD에 대해서는 ps-aux를 사용한다.
7. 문자렬이 인쇄된다. 2개의 감탄부호앞에 거꿀빗선이 있다는것을 주의해야 한다. 이것은 리력바꿔넣기를 보호하기 위해 필요하다.

2. 사용자입력읽기

$<변수 스크립트를 대화형으로 하기 위해 C쉘의 특수변수를 사용하여 표준입력을 변수에 읽어 들인다. $<는 표준입력으로부터 행바꾸기를 제외하고 그 행을 읽어 들이고 행을 변수에 대입한다.[6]

실례 9-67

(The Script - greeting)

[6]. 입력의 한개 행을 읽어 들이는 다른 방법은 setvariable = 'head –1'이다.

```
#/bin/csh  -f
# The greeting script
1. echo -n  "What is your name? "
2. set name = $<
3. echo Greetings to you, $name.

(The Command Line)
% chmod +x greeting
% greeting
1. What is your name?  Dan Savage
3. Greetings to you, Dan Savage.
```

설명

1. 문자렬이 현시장치에 출력된다. -n추가선택은 echo지령이 문자렬의 끝에서 행바꾸기를 삭제하게 한다. echo지령의 다른 판본들은 행바꾸기를 삭제하기 위해 \ c를 사용한다. 실례: echo hello\ c
2. 행바꾸기까지 말단에 입력되는 모든것이 name변수에 문자렬로 기억된다.
3. 변수바꿔넣기가 진행된후 문자렬이 인쇄된다.

입력문자렬로부터 단어목록만들기 $<변수로부터의 입력이 문자렬로 기억되므로 문자렬을 단어목록으로 나누어야 한다.

실례 9-68

```
1. % echo What is your full name\ ?
2. % set name = $<
   Lola Just-in Lue
3. % echo Hi $nanie[l]
   Hi Lola Justin Lue
4. % echo $name[2]
   Subscript out of range.
5. % set name = ( $nanie )
6. % echo Hi $name[l]
   Hi Lola

7. % echo $name[2] $name[3]
   Just-in Lue
```

설명

1. 사용자입력이 요구된다.
2. 특수변수 $<가 사용자로부터 문자렬형식으로 입력을 받는다.
3. 값 Lola JustinLue가 한개의 문자렬로 기억되므로 보조스크립트 [1]이 전체 문자렬을 현시한다. 보조스크립트는 1에서 시작한다.
4. 문자렬은 한개 단어로 이루어 진다. 2개의 단어가 없기때문에 보조스크립트 [2]을 사용할 때 쉘은 subscript is out of range를 통보한다.
5. 단어목록을 작성하기 위하여 문자렬을 괄호안에 넣는다. 한개 배렬이 작성

된다. 문자렬이 단어들의 목록으로 분리되여 변수 name에 대입된다.
6. 배렬의 첫번째 요소가 인쇄된다.
7. 배렬의 두번째, 세번째 요소가 인쇄된다.

3. 산수연산

쉘스크립트에서 수학문제들을 풀 필요는 실제로 없지만 때때로 순환계수기를 증가시키거나 감소시키는것을 위해 산수연산이 필요하다. C쉘은 옹근수산수연산만을 지원한다.

@기호를 사용하여 계산결과를 수변수에 대입한다.

산수연산자 표 9-8에 있는 다음의 연산자들을 사용하여 옹근수산수연산을 진행한다. 그것들은 C프로그람언어에 있는것과 같다. 연산자우선권을 표 9-8에 주었다. 또한 표 9-9에서 보여 준 지름대입연산자들은 C언어에서 넘어 온것들이다.

표 9-8. 연산자

기 능	연산자
더하기	+
덜기	-
나누기	/
곱하기	*
나머지	%
왼쪽자리옮김	<
오른쪽자리옮김	>

표 9-9. 지름연산

연산자	의미	등가식
+=	@num+=2	@num=$num+2
-=	@num-=4	@num=$num-4
*=	@num*=3	@num=$num*3
/=	@num/=2	@num=$num/2
++	@num++	@num=$num+1
--	@num--	@num=$num-1

실례 9-69

```
1.  % @ sum =4+6
    echo $sum
    10

2.  % @ SUBI++
```

```
echo $suin
11
```

3. % **@ sum += 3**
   echo $sum
   14

4. % **@ SUBl--**
   echo $sum
   13

5. % **@ n = 3+4**
   @: Badly formed number

설명

1. 변수 sum에 4와 6을 더한 결과가 대입된다(**@**다음에 공백이 요구된다.).
2. 변수 sum이 1만큼 증가된다.
3. 변수 sum이 3만큼 증가된다.
4. 변수 sum이 1만큼 감소한다.
5. **@**기호다음과 연산자주위에 공백이 있어야 한다.

류동소수점산수연산 류동소수점산수연산은 이 쉘에 의해 지원되지 않으므로 더 복잡한 수학적연산을 할 필요가 있으면 UNIX편의프로그람들을 사용할수 있다. 복합계산을 할 때 bc와 nawk를 쓸수 있다.

실례 9-70

(The Command Line)

1. **set n='echo "scale=3; 13 / 2" | bc'**
   echo $n
   6.500

2. set products'**nawk -v x=2.45 -vy3.124** "BEGIN{\
   printf "%.2f\ n", x * y }'•
   % **echo $product**
   7.65

설명

1. echo지령의 출력이 bc프로그람으로 파이프된다. scale이 3으로 설정된다. 즉 소수점아래 중요한 3개 수자들이 인쇄된다. 계산은 13을 2로 나누는것이다. 전체 파이프행은 거꿀인용부호안에 닫긴다. 지령바꿔넣기가 진행되고 출력이 변수 n에 대입된다.
2. nawk프로그람은 지령행에서 넘겨 진 인수목록으로부터 값을 받는다. nawk에 넘겨 진 매 인수는 그앞에 –v추가선택이 붙는다. 실례로 –v x=2.45, –v y=3.124수들이 곱해 진 다음 printf함수는 그 결과를 소수점아래 두자리까지 형식화하여 인쇄한다. 그 출력은 변수 product에 대입된다.

4. 스크립트의 오유수정

C쉘스크립트는 흔히 간단한 문법적오유나 론리적오유때문에 실패한다. csh지령에 대한 추가선택들이 프로그람오유수정을 도와 준다(표 9-10).

표 9-10. Echo(-x)와 verbose(-v)

csh에 대한 추가선택	
csh-x scriptname	변수바꿔넣기후 실행하기전에 스크립트의 매행을 현시한다.
csh-v scrptname	스크립트의 매행을 그것을 입력한것대로 실행전에 현시한다.
csh-n scriptname	지령들을 실행하지는 않고 해석만 한다.
set지령에 대한 인수	
set echo	지령바꿔넣기후 실행전에 스크립트의 매행을 현시한다.
set verbose	스크립트의 매행을 그것을 입력한것대로 실행전에 현시한다.
스크립트의 첫번째 행	
#!/bin/csh-xv	echo와 verbose를 둘다 선택한다. 이 추가선택들은 개개로 호출되거나 다른 csh호출인수들과 함께 조합될수 있다.

실례 9-71

(The *-v and -x* Options)

1. % **cat practice**
   #!/bin/ash
   echo Hello $LOGNAMB
   echo The date is 'date'
   echo Your home shell is $SHELL
   echo Good-bye $LOGNAME

2. % **csh -v practice**
   echo Hello $LOGNAME
   Hello ellie
   echo The date is 'date'
   The date is Sun May 23 12:24:07 PUT   2001
   echo Your log-in shell is $ SHELL
   Your login shell is /bin/csh
   echo Good-bye $LOGNAME
   Good-bye ellie

3. % **csh -x practice**
   echo Hello ellie
   Hello ellie
   echo The date is 'date'
   date
   The date is Sun May 23 12:24:15 PUT   2001
   echo Your login shell is /bin/csh

Your login shell is /bin/csh
echo Good-bye ellie
Good-bye ellie

설명

1. C쉘스크립트의 내용들이 현시된다. 변수와 지령바꿔넣기행들로부터 echo와 verbose가 어떻게 다른가를 알수 있다.
2. csh지령에 대한 -v추가선택은 verbose수단을 쓸수 있게 한다. 스크립트의 매행이 입력된대로 현시되고 그다음 행이 실행된다.
3. csh지령에 대한 -x추가선택은 echoing을 할수 있게 한다. 스크립트의 매행은 변수와 지령바꿔넣기가 진행된후 현시되고 그다음에 행이 실행된다. 이 수단을 사용하여 지령과 변수바꿔넣기의 결과로 무엇이 교체되는가를 검사할수 있으므로 verbose추가선택보다 더 자주 사용된다.

실례 9-72

(Echo and Verbose)

1. % **cat practice**
   #!/bin/csh
   echo Hello $LOGNAME echo
   The date is 'date'
   set echo
   echo Your home shell is $ SHELL unset
   echo echo Good-bye $LOGNAME
   % chmod +x practice
2. % **practice**
   Hello ellie The date is Sun May 26 12:25:16 PDT 2001

--> echo Your login shell is /bin/csh
--> Your login shell is /bin/csh
--> unset echo
Good-bye ellie

설명

1. echo추가선택이 스크립트안에서 설정되고 해제된다. 이것은 전체 스크립트의 매행을 출력하지 않고 잘 실행되지 않는 스크립트의 어떤 부분들을 오유수정할수 있게 한다.
2. - ->는 표시가 설정된것을 표식한다. 매행은 변수와 지령바꿔넣기가 진행된 후 인쇄되고 그다음 실행된다.

실례 9-73

1. % **cat practice**

```
#!/bin/csh
echo Hello $LOGNAUE
echo The date is 'date'
set verbose
echo Your home shell is $SHBLL
unset verbose
echo Good-bye $LOGNAME

2.  % practice
    Hello ellie The date is Sun May 23 12:30:09 PDT   2001
--> echo Your login shell is $SHBLL
--> Your login shell is /bin/csh
--> unset verbose Good-bye ellie
```

설명

1. verbose추가선택이 스크립트안에서 설정되고 해제된다.
2. - ->는 verbose가 선택된 곳을 표시한다. 행들은 스크립트에 입력된대로 인쇄되고 그다음에 실행된다.

5. 지령행인수

쉘스크립트들은 지령행인수들을 가질수 있다. 인수들은 프로그람의 동작을 다르게 수정하는데 사용된다. C쉘은 지령행인수들을 위치파라메터들에 대입하는데 대입될수 있는 인수들의 수에는 제한이 없다(Bourne쉘은 위치파라메터의 개수를 9개로 제한한다.). 위치파라메터들은 수변수들이다. 스크립트의 이름이 $0에 대입되고 스크립트이름 다음에 오는 단어들이 $1, $2, $3, ... ${10}, ${11 등에 대입된다. $1은 첫번째 지령행인수이다.

위치파라메터사용외에도 C쉘은 내부지령 argv배렬을 제공한다.

위치파라메터들과 argv 만일 argv배렬표시를 사용하면 유효보조스크립트는 지령행으로부터 넘겨 지는 인수에 대응하도록 제공되여야 한다. 그렇지 않으면 오유통보 subscript out of range가 C쉘에 의해서 보내진다. argv배렬은 스크립트이름을 포함하지 않는다. 첫번째 인수는 $argv[1]이고 인수들의 수는 $#argv로 표시된다(인수들의 수를 현시하는 다른 방법은 없다.). 지령행인수들의 목록을 표 9-11에 보여 주었다.

표 9-11. 지령행인수

인 수	의 미
$0	스크립트의 이름
$1,$2...${10}...	첫번째, 두번째 위치파라메터들이 표식 $가 앞에 붙은 수에 의해 참조된다. 대괄호는 수자 10을 대괄호에 넣으므로 첫번째 위치파라메터를 표시하지 않게 한다.

(표계속)

인 수	의 미
$*	모든 위치파라메터들
$argv[0]	유효값이 아니다. 아무것도 인쇄되지 않는다. C쉘의 배렬의 첨수는 1에서 시작한다.
$argv[1] $argv[2]...	첫번째 인수, 두번째 인수...
$argv[*]	모든 인수들
$argv	모든 인수들
$#argv	인수들의 개수
$argv[$#argv]	마지막인수

실례 9-74

(The Script)

```
#!/bin/csh -f
# The greetings script
# This script greets a user whose name is typed in at the
# command line.

1. echo $0 to you $1 $2 $3
2. echo Welcome to this day 'date | awk '{print $1, $2, $3}
3. echo Hope you have a nice day, $argv[1]\ '
4. echo Good-bye $argv[1] $argv[2] $argv[3]
```

(The Command Line)

```
% chmod +x greetings

% greetings Guy Quigley
1. greetings to you Guy Quigley
2. Welcome to this day Fn Aug 28.
3. Hope you have a nice day. Guy'
4. Subscript out of range
```

설명

1. 스크립트의 이름과 첫번째 3개 위치파라메터들이 현시되게 된다. 지령행으로부터 2개의 위치파라메터들 Guy과 Quigley만이 입력되므로 $1은 Guy로, $2는 Quigley으로 되고 $3은 정의되지 않는다.
2. awk지령이 단일인용부호속에 있으므로 쉘은 awk마당번호인 $1, $2, $3을 위치파라메터들로 혼돈하지 않는다(awk의 마당지시자들인 $1, $2, $3을 쉘의 위치파라메터들로 혼돈하지 말아야 한다.).
3. argv배렬에 지령행으로부터 들어 오는 값들이 대입된다. guy는 argv[1]에 대입되고 그 값이 현시된다. argv배렬을 사용하여 스크립트안에서 지령행

인수들을 표시하거나 위치파라메터를 사용할수 있다. 그 차이는 값을 가지지 않는 위치파라메터를 참조할 때 그것이 오유를 발생시키지 않는데 이와는 달리 대입되지 않는 argv값은 스크립트가 오유통보subscript out of rangeer와 함께 탈퇴하게 하는것이다.

4. 쉘은 argv[3]에 대한 값이 없기때문에 오유 Subscript out of range를 인쇄한다.

6. 흐름조종과 조건구성체

판단할 때 if, if/else, if/else if와 swith지령들을 사용한다. 이 지령들은 식이 참인가 거짓인가에 따라서 판단하게 함으로써 프로그람의 흐름을 조종한다.

식시험 식은 연산자들로 분리된 연산항들의 모임으로 이루어 진다. 연산자들과 그 우선권을 표 9-12와 9-13에 보여 준다. 식을 검사하기 위해형식을 괄호안에 넣는다. C쉘은 식을 평가하여 0 또는 0이 아닌 수값을 내보낸다. 만일 그 결과가 0이 아니면 식은 참이고 결과가 0이면 식은 거짓으로 된다.

표 9-12. 비교연산자와 론리연산자

연산자	의 미	실 례
==	...과 같다.	$x==$y
!=	...과 같지 않다.	$x!=$y
>	...보다 크다.	$x>$y
<	...보다 작다.	$x>=$y
>=	...보다 크거나 같다.	$x<$y
<=	...보다 작거나 같다.	$x<=$y
=~	문자렬이 일치된다.	$ans=~[Yy]*
!~	문자렬이 일치되지 않는다.	$ans!~[Yy]*
!	론리부정	!$x
||	론리더하기	$x11$y

론리곱하기(&&)를 사용하여 식을 평가할 때 쉘은 왼쪽에서 오른쪽으로 평가한다. 첫번째 식(&&의 앞에 있는)이 거짓이면 쉘은 전체 식의 결과로서 거짓을 대입하고 나머지식을 검사하지 않는다. 만일 첫번째 식이 거짓이면 론리곱하기(&&)연산자를 사용할 때 전체 식은 거짓으로 된다. 론리연산자 &&량옆에 있는 식이 둘다 참으로 될 때 전체 식이 참으로 된다. 론리더하기(||)로 식을 평가할 때 ||의 왼쪽에 있는 첫번째 식이 참이면 쉘은 전체 식의 결과를 참으로 하고 나머지식을 검사하지 않는다. 론리더하기식에서 한개의 식만이 참으로 되면 전체 결과가 참으로 된다. 론리부정은 일항연산자이다. 즉 그것은 한개식을 평가한다. 만일 부정연산자의 오른쪽에 있는 식이 참이면 그 식은 거짓으로 된다. 만일 거짓이면 그 식은 참으로 된다.

우선권과 결합성 C언어에서와 같이 C쉘은 식을 검사할 때 우선권과 결합성을 사

용한다. 만일 식이 다음과 같이 여러가지의 연산자들로 이루어 진다면

```
@ x = 5 + 3 * 2
echo $x
11
```

쉘은 연산자들을 정해 진 순서로 읽는다. 우선권은 어느 연산자가 제일 중요한가를 표시한다. 결합성은 우선권이 같을 때 쉘이 식을 왼쪽으로부터 오른쪽으로 읽는가 또는 오른쪽으로부터 왼쪽으로 읽는가를 표시한다. 연산식(쉘스크립트들에서 필요되지 않는)에서와는 달리 결합성의 순서는 우선권이 같다면 왼쪽으로부터 오른쪽으로 읽는다. 괄호를 사용하여 그 순서를 변화시킬수 있다(표 9-13을 참고).

```
@ x = ( 5 + 3 ) * 2
echo  $x
16
```

식은 수, 관계식 또는 론리식일수 있다. 수식은 다음과 같은 산수연산자들을 사용한다.

```
+ - * / ++ -- %
```

관계식은 참(령이 아닌) 또는 거짓(0)결과를 내는 연산자들을 사용한다.

```
> < >= <= == !=
```

론리식은 다음과 같은 연산자들을 사용한다.

```
!  &&  ||
```

표 9-13. 연산자우선권

우선권	연산자	의 미
높다	( )	우선권을 변화시킨다.
	~	보수
	!	론리부정
	* / %	곱하기, 나누기, 나머지
	+ -	더하기, 덜기
	<< >>	왼쪽, 오른쪽 비트자리옮김
	> >= < <=	관계연산자들:보다 크다. 보다 작다.
	== !=	같기: 같다. 같지 않다.
	=~ !~	패턴일치: 일치된다. 일치되지 않는다.
	&	비트론리적
	^	비트배타적론리합
	|	비트론리합
	&&	론리곱하기
낮다	||	론리더하기

if명령문 가장 간단한 형식의 조건명령은 if명령이다. if가 검사된후 식이 참으로 되면 예약어 then다음의 지령들이 endif가 나타날 때까지 실행된다. 예약어 endif는 블로크를 끝낸다. 매개의 if명령이 endif로 끝날 때 if명령은 겹싸여 질수 있다. Endif는 가장 가까운 if와 일치된다.

형식

```
if ( expression ) then
    지령
    지령
endif
```

실례 9-75

(In the Script: Checking for Arguments)

```
1.  if ( $#argv != 1 ) then
2.          echo "$0 requires an argument"
3.          exit 1
4.  endif
```

설명

1. 이 행은 다음과 같다. 지령행으로부터 넘겨 진 인수들의 수($#argv)가 1과 같지 않다면...
2. 만일 첫번째 행이 참이면 이 행과 3행이 실행된다.
3. 프로그람은 값을 1로 하고 탈퇴한다. 이것은 실패했다는것을 의미한다.
4. 매 if블로크는 endif명령으로 끝난다.

검사와 해제, Null변수 특수변수 $?는 변수가 설정되였는가를 검사하는데 사용된다. 변수가 빈값으로 설정되면 참을 되돌린다.

실례 9-76

(From .cshrc File)

```
if ( $?proinpt ) then
set history =32
end if
```

설명

새로운 csh프로그람을 시작할 때마다 .cshrc파일이 실행된다.
$?는 변수가 설정되였는가를 검사하는데 사용된다. 이 실례에서 쉘은 재촉문이 설정되였는가를 검사한다. 재촉문이 설정되면 스크립트가 아니라 대화형쉘이 실행된다. 재촉문은 대화형을 위해서만 설정된다. 리력기구는 대화형으로 실행될 때만 사용할수 있으므로 쉘은 스크립트를 실행할 때 리력을 설정하지 않는다.

실례 9-77

(The Script)

```
        echo -n "What is your name?
    1.  set name = $<
    2.  if ( "$name" != "" ) then
              grep "$name" datafile
        endif
```

설명

1. 사용자입력이 요구된다. 만일 사용자가 Enter건을 누르면 변수 name이 설정되지만 그것은 빈값으로 설정된다.
2. 사용자가 name에 한개이상의 단어를 입력할 때 식을 평가할수 있도록 하기 위해서 변수는 2중인용부호속에 넣는다. 만일 인용부호를 없애고 사용자가 첫번째와 마지막이름을 입력했다면 쉘은 오유통보 if:Expression syntax를 통보하고 스크립트를 탈퇴한다. 빈 2중인용부호는 빈값문자렬을 현시한다.

if/else 명령문 if/else지령은 2중가지 조종구조이다. 만일 if지령다음에 있는 식이 참이면 그다음에 오는 블로크가 실행된다. 그렇지 않으면 else다음의 블로크가 실행된다. endif는 가장 안쪽의 if를 끝내고 그 명령을 완료한다.

형식

```
if ( expression ) then
   지령
else
  지령
endif
```

실례 9-78

```
1.  if ( $answer =~ [Yy]* ) then
2.          mail bob < message
3.  else
4.            mail John < datafile
5.  endif
```

설명

1. 이 행은 다음과 같다. $answer의 값이 Y나 y로 시작되여 그뒤에 임의의 수의 문자들이 오는것과 같으면 행 2로 간다. 그렇지 않으면 행 3으로 간다 (별표는 쉘메타문자이다).
2. 사용자 bob에 파일 message의 내용들을 보낸다.
3. 행 1이 참이 아니면 else다음의 지령들이 실행된다.
4. 사용자 john에 파일 datafile의 내용들이 전송된다.
5. endif는 If블로크를 끝낸다.

식오유수정 C쉘에 대한 -x추가선택은 실행할 때 스크립트에서 무엇이 진행되는가를 추적할수 있게 한다. 만일 무엇이 진행되고 있는가를 확신할수 없을 때 이것은 스크

립트를 오유수정하는 좋은 방법으로 된다.

실례 9-79

(The Script: Using Logical Expressions ar.a Checking Values)
```
    #!/bin/csh -f
    # Scriptname: logical
    set x = 1
    set y = 2
    set z = 3
1.  if ( ( "$x" && "$y" ) ||  ! "$z" ) then
      # Note: grouping and parentheses
2.      echo TRUE
    else
        echo FALSE
    endif
```
(The Output)
```
3.  % csh -x logical
    set x = 1
    set y = 2
    set z = 3
    if  ( ( 1 && 2 ) || ! 3 ) then
    echo TRUE
    TRUE
    else
    %
```

설명

1. 론리식이 평가된다. 첫번째 식은 괄호속에 놓인다(&&가 ||보다 더 높은 우선권을 가지기때문에 필요하지 않다.). 괄호는 겹싸여 질 때 공백을 요구하지 않지만 부정연산자(!)다음에 공백이 있어야 한다.
2. 식이 참으로 되면 이 행이 실행된다.
3. csh프로그람은 –x추가선택을 선택하여 실행한다. 이것은 echoing을 설정한다. 스크립트의 매행은 변수바꿔넣기가 진행된후 다시 출력된다.

명령과 단일지령 식다음에 단일지령이 오면 예약어 then과 endif가 필요 없다.

형식

if (expression)단일지령

실례 9-80

```
if (s#argv == 0) exit 1
```

설명

식이 검사된다. 만일 지령행인수들의 수 $#argv가 0과 같으면 프로그람은 상

태를 1로 하고 탈퇴한다.

if/else if명령문 if/else if구조는 다중판단을 할수 있게 한다. 몇개의 식들이 검사될수 있는데 평가되는 한개의 식이 참이면 그다음에 오는 명령블로크가 실행된다. 만일 어느 한 식도 참이 아니면 else블로크가 실행된다.

형식

```
if ( expression ) then
    지령
    지령
else if ( expression ) then
    지령
    지령
else
    지령
endif
```

실례 9-81

```
(The Script: grade)
     #!/bin/csh -f
     # This script is called grade
     echo -n "What was your grade? °
     set grade = $<
1.   if ( $grade >= 90 && $grade <» 100 ) then
          echo "You got an A\ !"
2.   else if ( $grade > 79 ) then
          echo "You got a B"
3.   else if ( $grade > 69 ) then
          echo "You're average"
     else
4.        echo "Better study"
5.   endif
```

설명

1. 만일 grade가 90보다 크거나 작고 100보다 작거나 같으면 You got an A!가 인쇄된다. &&의 량옆의 식들이 다 참이여야 한다. 그렇지 않으면 프로그람조종이 2행에 있는 elseif로 넘어 간다.
2. 행 1이 거짓이면 식(행2)을 검사하고 그것이 참이면 you got a B가 인쇄된다.
3. 행 1과 2가 둘다 거짓이면 이 행을 실행한다. 만일 이 식이 참이면 echo You're average가 인쇄된다.
4. 우의 모든 식들이 거짓으로 검사되면 else블로크안에 있는 명령들이 실행된다.

5. endif는 전체 if구조를 끝낸다.

탈퇴상태와 상태변수 지령이 성공했으면 매 UNIX지령은 탈퇴상태를 0으로 되돌린다. 만일 지령이 실패했으면 그것은 0이 아닌 탈퇴상태를 되돌린다.

C쉘상태변수의 값을 보고 지령이 성공했는가 또는 실패했는가를 검사할수 있다.

상태변수는 실행된 마지막지령의 탈퇴상태를 가진다.

실례 9-82

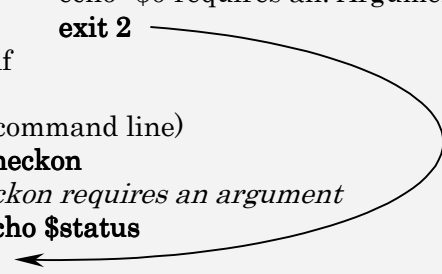
```
1.  % grep ellie /etc/passwd
      ellie:pHAZk66gA:9496:41:Ellie:/home/jody/ellie:/bin/ash
2   % echo $status
    0                          # Zero shows that grep was a success

3.  % grep joe /etc/passwd

4.  % echo $status
     1                         # Nonzero shows that grep tailed
```

설명

1. grep프로그람이 /etc/passwd파일에서 ellie를 찾았다.
2. grep프로그람이 패턴 ellie를 찾는다면 탈퇴할 때 0상태를 되돌린다.
3. grep프로그람은 /etc/passwd파일에서 joe는 찾지 못했다.
4. grep프로그람은 패턴이 발견되지 않으면 0이 아닌 상태를 되돌린다.

쉘스크립트로부터 탈퇴 쉘스크립트에서 exit지령을 사용하여 쉘재촉문을 다시 현시할수 있다.

exit지령은 탈퇴형을 표시하는 옹근수값을 가진다. 령이 아닌 인수는 실패를 현시하고 령은 성공을 현시한다. 그 수는 0부터 255사이여야 한다.

실례 9-83

```
(The checkon Shell Script)
    #l/bin/csh -f
1.  if ( $#argv != 1 ) then
2.          echo "$0 requires an. Argument"
3.          exit 2
4.  endif

(At the command line)
5.  % checkon
    checkon requires an argument
6.  % echo $status
    2
```

설명

1. 지령행으로부터 넘겨 진 인수들의 수가 1이 아니면 행 2로 간다.

2. echo지령은 스크립트이름($0)과 문자렬 requires an argument를 인쇄한다.
3. 프로그람은 값을 2로 하여 재촉문으로 다시 탈퇴한다. 이 값은 어미쉘의 status변수에 기억된다.
4. 조건 if의 끝이다.
5. 지령행에서 프로그람 checkon이 인수없이 실행된다.
6. 프로그람은 값을 2로 하여 탈퇴하는데 그 값은 status변수에 기억된다.

스크립트에서 상태변수사용 스크립트에서 status변수를 사용하여 지령의 상태를 검사할수 있다. status변수에 실행된 마지막지령의 값이 대입된다.

실례 9-84

```
(The Script)
    #'./bin/csh –f
1.  ypmatch   $1   passwd >& /dev/null
2.  if ( $status == 0 ) then
3.          echo Found $1 in the NIS database
    endif
```

설명

1. ypmatch프로그람은 NIS자료기지를 검사하여 첫번째 인수로 넘겨 진 사용자의 이름이 자료기지에 있는가를 본다.
2. 마지막지령으로부터 넘겨 진 status가 0이면 then블로크가 실행된다.
3. if검사식이 참으로 평가됐으면 이 행이 실행된다.

조건부에서 지령평가 C쉘은 조건부에 있는 식을 평가한다. 조건부에서 지령을 평가하기 위해서 지령을 대괄호속에 넣어야 한다. 만일 지령이 성공하면 즉 령탈퇴상태를 넘기면 대괄호는 쉘이 식을 참(1)으로 평가하도록 한다.[8]

만일 지령이 실패하면 탈퇴상태는 령이 아니고 식은 거짓(0)으로 평가된다.

조건부에서 지령을 사용할 때 그 지령의 탈퇴상태를 아는것이 중요하다.

실례로 grep프로그람은 탐색하는 패턴을 찾으면 탈퇴상태를 0으로 넘기고 패턴을 찾지 못하면 1로 넘기며 파일을 찾지 못하면 2로 넘긴다. awk와 sed가 패턴들을 탐색할 때 패턴탐색에서 성공했는가 안했는가에 따라 0을 되돌린다.

awk와 sed가 성공했는가는 문법이 정확한가 아닌가에 달려 있다. 즉 지령을 정확히 입력했으면 awk와 sed의 탈퇴상태는 0이다. 만일 감탄부호가 식앞에 놓이면 그것은 전체 식을 부정하여 그것이 참일 때는 거짓으로 되고 거짓일 때에는 참으로 된다. 감탄부호다음에 공백을 꼭 두어야 한다. 그렇지 않으면 C쉘은 리력기구를 호출한다.

형식

```
if  {  ( command )  }   then
      지령
      지령
endif
```

---

8. 쉘에 의해서 전환되여 식이 참 또는 거짓결과를 나타낸다.

실례 9-85

```
#!/bin/csh -f
1.  if  {( who | grep $1 >& /dev/null )} then
2.        echo $1 is logged on and running:
3.        ps -ef | grep "" *$1"          # ps -aux for BSD
4.  endif
```

설명

1. who지령은 grep지령에 파이프된다. 모든 출력은 UNIX"비트바키트"인 /dev/null에 보내진다. who지령의 출력은 grep로 보내진다. grep는 $1변수에 기억된 사용자의 이름을 탐색한다(첫번째 지령행인수). 만일 grep가 성공하여 사용자를 찾으면 0탈퇴상태가 복귀된다. 쉘은 그다음에 grep지령의 탈퇴상태를 1(참)로 전환한다. 쉘은 식이 참이라고 평가되면 then과 endif사이의 지령들을 실행한다.
2. C쉘이 행 1에 있는 식이 참이라고 평가하면 행 2와 행 3이 실행된다.
3. 실행되는 모든 프로쎄스들과 $에 의한 모든 프로쎄스들이 현시된다.
4. endif는 if명령을 끝낸다.

형식

```
if ! {(command)}then
```

실례 9-86

```
1.  if  !  {( ypmatch $user passwd >& /dev/null )} then
2.        echo $user is not a user here.
      exit 1
3.  endif
```

설명

1. 망을 사용하려면 ypmatch지령을 사용하여 NIS passwd파일을 탐색한다. 지령이 passwd파일에서 사용자를 찾으면 식이 참으로 된다. 식앞에 있는 감탄부호(!)는 그 식을 부정하거나 반전시킨다. 즉 그것이 참이면 거짓으로 되게 하고 거짓이면 참으로 되게 한다.
2. 만일 식이 참이 아니면 사용자는 발견되지 않고 이 행이 실행된다.
3. endif는 이 if블로크를 끝낸다.

goto goto는 프로그람에서 어떤 표식까지 뛰여 넘어 그 점에서 실행을 시작하게 한다. 많은 프로그람작성자들이 goto를 사용하기 좋아 하지 않지만 그것들은 때때로 겹쌓인 순환에서 벗어 나는데 쓸모가 있다.

실례 9-87

```
(The Script)
#'./bin/csh -£
```

```
1.   startover;
2.   echo "What was your grade?
     set grade = $<
3.   if ( "$grade" < 0   | "$grade" > 100 ) then
4.        echo "Illegal grade"
5.        goto startover

     endif
     if ( $grade >= 8.9 ) then
          echo "A for the genius\ !"
     else if ( $grade >= 79 ) then
          .. < Program continues >
```

설명

1. 표식은 옹근두점이 붙은 사용정의된 단어이다. 이 표식은 startover라고 부른다. 프로그람이 실행되는 동안 쉘이 그 표식으로 넘어 가도록 명백히 지정되지 않으면 표식은 쉘에 의해서 무시된다.
2. 사용자입력이 요구된다.
3. 만일 식이 참이면(사용자는 0보다 작거나 100보다 큰 grade를 입력하였다.)문자 illegal grade가 인쇄되고 표식 startover에서 goto가 실행을 시작한다. 프로그람은 그 점으로부터 실행을 계속한다.
4. if식이 거짓으로 검사되여 이 행이 인쇄된다.
5. goto는 조종을 1행에로 넘기고 표식 startover다음에서 실행이 시작된다.

파일시험 C쉘에는 그것이 등록부인가, 보통 파일인가(등록부가 아닌가) 또는 읽기가능한 파일인가 등과 같은 파일속성들을 검사하기 위한 추가선택들의 모임이 내장되여 있다.

다른 형태의 파일시험로서 UNIX의 test지령이 사용된다.

파일시험을 위한 내부추가선택들을 표 9-14에 보여 준다.

표 9-14. 파일시험

검사기발	다음의 경우 참으로 된다.
-r	현재사용자가 파일을 읽을수 있다.
-w	현재사용자가 파일에 쓸수 있다.
-x	현재사용자가 파일을 실행할수 있다.
-e	파일이 존재한다.
-o	현재사용자가 파일을 수집한다.
-z	파일의 길이가 0이다.
-d	파일은 등록부이다.
-f	보통파일이다.

실례 9-88

```
#!/bin/csh -f
1.  if ( -e file ) then
          echo file exists
     endif
2.  if ( -d file ) then
          echo file is a directory
     endif
3.  if ( ! -2 file ) then
          echo file is not of zero length
     endif
4.  if ( -r file && -w file ) then
          echo f-le is readable and writeable
     endif
```

설명

1. 명령은 다음과 같다. 만일 파일이 존재하면...
2. 명령은 다음과 같다. 만일 파일이 등록부이면...
3. 명령은 다음과 같다. 파일의 길이가 0이 아니면...
4. 명령은 다음과 같다. 파일이 읽기, 쓰기가능하면...

   파일시험기발들은 -rwx파일에서와 같이 축적될수 없다. 파일이름앞에 한개의 추가선택만이 놓인다(실례 -r파일 && -w파일 &&-x파일).

test지령과 파일시험 UNIX의 test지령에는 C쉘에 내부추가선택들뿐아니라 C쉘에 내장되여 있지 않는 많은 추가선택들이 있다. 표 9-15에 검사추가선택들의 목록을 보여주었다. 블로크와 특수문자파일들 즉 setuid파일과 같은 파일들의 덜 일반적인 속성들을 검사할 때 이러한 추가적인 추가선택들이 필요할 때가 있다. test지령은 식을 평가하여 성공이면 0을, 실패이면 1로 탈퇴상태를 되돌린다. if조건명령안에서 test지령을 사용할 때 그 지령을 대괄호속에 넣어서 쉘이 탈퇴상태를 정확히 평가할수 있도록 하여야 한다.[9]

조건명령에서 test지령을 사용하려면 C쉘이 탈퇴상태를 정확히 평가하도록 다른 지령들에 사용하던것과 같이 대괄호를 사용하여야 한다.

표 9-15. test지령으로 파일시험

추가선택	다음의 경우에 참으로 검사된다.
-b	파일이 블로크특수파일이다.
-c	파일이 문자특수파일이다.
-d	파일이 존재하며 등록부파일이다.

9. 일반적인 오유는 스크립트를 test라고 이름 짓는것이다. 만일 탐색경로의 처음에 UNIX test지령이 있으면 그것을 실행시킨다. test지령은 오유를 현시하거나 문법이 정확하면 아무것도 현시하지 않는다.

(표계속)

추가선택	다음의 경우에 참으로 검사된다.
-f	파일이 존재하며 보통파일이다.
-g	파일은 집단설정식별IDbit모임을 가진다.
-k	파일은 스틱키비트모임을 가진다.
-p	파일의 이름은 파이프이다.
-r	현재 사용하는 파일을 읽을수 있다.
-s	파일이 존재하면서 빈 파일이 아니다.
-tn	n은 말단에 대한 파일서술자이다.
-u	파일은 사용자설정식별IDbit모임을 가진다.
-w	현재사용자는 파일에 쓸수 있다.
-x	현재사용자는 파일을 실행시킬수 있다.

실례 9-89

1. **if { test –b file }** echo file is a block device file
2. **if { test –u file }** echo file has the set-usr-id bit set

설명

1. 명령은 다음과 같다. 만일 파일이 블로크특수파일이면(/dev에 있는)
2. 명령은 다음과 같다. 만일 파일이 setuid프로그람(사용자설정식별번호)이면...

조건명령겹싸기 조건명령들은 겹싸여 질수 있다. 매 if는 대응하는 elseif를 가져야 한다(elseif는 endif를 가지지 않는다). 프로그람을 더 효률적으로 읽고 검사하기 위하여 겹싸여 진 명령들을 행에서 같은 길이만큼 들여다 써서 if와 endif들을 행들에서 같은 위치에 놓는것이 좋다.

실례 9-90

(The Script)

```
#!/bin/csh -f
# Scriptname: filecheck
# Usage: filecheck filename
set file=$1
```

```
1┌─if ( ! -e $file ) then
 │        echo "$file does not exist"
 │        exit 1
 └─endif
2┌─if ( -d $file ) then
 │        echo "Sfile is a directory"
3│ else if <-f $file) then
4│    ┌─ if ( -r $file && -x $file ) then        # Nested if construct
 │    │    echo "You have read and execute permission on. $file"
5│    └─ endif
 │ else
 │      print "$file is neither a plain file nor a directory."
6└─endif
```

(The Command Line)

$ **filecheck testing**

You have read and execute permission of file testing.

설명

1. 파일(변수바꿔넣기후)이 존재하지 않는 파일이면 예약어 then다음의 지령들이 실행된다. 출력값이 1이면 프로그람이 실패했다는것이다.
2. 파일이 등록부이면 testing is a directory를 인쇄한다.
3. 파일이 등록부가 아니면 그리고 파일이 보통파일이면 … 다음명령이 실행된다.
4. 이 if는 그전의 if안에 겹싸여 진다. 만일 파일을 읽을수도 있고 쓸수도 있고 실행할수도 있으면 … 이 if는 자기의 endif를 가지며 행의 같은 위치에 놓여 그것이 어디에 속했는가를 나타낸다.
5. endif는 제일 안쪽의 if구조를 끌낸다.
6. endif는 제일 바깥 if구조를 끌낸다.

switch지령 switch지령은 if-then-elseif구조를 사용하는 다른 방법이다. 때때로 switch지령은 다중추가선택들을 처리할 때 프로그람을 더 명백히 읽을수 있게 한다.

switch식에서 값은 예약어 case다음에 오는 표식이라고 하는 식들과 비교된다.

case표식들은 상수값과 통용기호로 될수 있다. 표식은 옹근두점으로 끝난다.

default표식은 선택적이지만 case표식들중 어느 하나도 switch식과 맞지 않으면 그것이 실행된다. breaksw을 사용하여 실행을 endsw로 넘긴다. 만일 breaksw가 생략되고 표식이 일치되면 일치된 표식다음의 명령들이 breaksw나 endsw가 나타날 때까지 실행된다.

형식

```
switch (variable)
case constant:
      지령
```

```
breaksw
case constant:
        지령
        breaksw
endsw
```

실례 9-91

```
(The Script - colors)
       #!/bin/csh -f
       # This script is called colors
 1.  echo -n "Which color do you like? "
 2.  set color = $<
 3.  switch ("$color")
 4.  case bl*:
             echo I feel $color
             echo The sky is $ col or
 5.    breaksw
 6.  case red;          # Is is red or is it yellow?
 7.  case yellow:
 8.          echo The sun is sometimes $color.
 9.          breaksw
10.  default:
11.          echo $color not one of the categories.
12.          breaksw
13.  endsw

(The Command Line)
% colors

(The Output)
1  Which color do you like? red
8  The sun is sometimes red.
1  Which color do you like? Doesn't matter
11 Doesn't matter is not one of the categories.
```

설명

1. 사용자입력이 요구된다.
2. 입력이 color변수에 대입된다.
3. switch명령이 변수를 평가한다. 사용자가 한개이상의 단어를 입력했을 때 변수는 2중인용부호속에 넣어 진다. switch명령은 한개의 단어를 평가하거나 단어들의 문자렬이 2중인용부호속에 있으면 그 단어들이 문자렬을 평가한다.
4. case표식은 bl*인데 switch식이 bl로 시작되는 임의의 문자들의 모임과 일치된다. 만일 사용자가 blue, black, blah, blast등을 입력하면 이 case표식다음의 지령들이 실행된다.

5. breaksw는 프로그람조종을 endsw명령에로 넘긴다.
6. 만일 switch명령이 표식 red와 일치되면 프로그람이 행 9에 있는 breaksw가 나타날 때까지 명령실행을 시작한다. 행 8이 실행된다. The sun is sometimes red가 현시된다.
7. 만일 행 4가 일치되지 않으면 표식 red와 yellow가 검사된다.
8. 만일 표식 red나 yellow가 일치되면 이 행이 실행된다.
9. breaksw는 프로그람조종을 endsw명령에로 넘긴다.
10. 추가선택표식들중 그 어느것도 switch식과 일치되지 않으면 기정표식에로 넘어 간다. 이것은 if/else if/else구조와 같다.
11. 사용자가 입력한것이 우의 표식들에서 어느 하나와도 일치되지 않으면 이 행이 인쇄된다.
12. 여기서 switch가 끝나므로 breaksw를 선택한다. 만일 후에 더 많은 표식들이 추가될 때 breaksw가 무시되지 않도록 여기에 breaksw를 쓰는것이 좋다.
13. endsw는 switch명령문을 끝낸다.

switch겹싸기 switch들은 겹싸여 질수 있다. 즉 switch명령과 그 추가선택들은 다른 switch명령문안에 한개의 표식으로 포함될수 있다. endsw로 매 switch명령을 끝내야 한다. default 표식들은 요구되지 않는다.

실례 9-92

```
(The Script: systype)
#!/bin/csh -f
# This script is called systype
# Program to determine the-type of system you are on.
echo “Your system type is: “
1.  set release =('uname -r')
2.  switch ('uname -a')
3.  case SunOS:
4.          switch (“$release”)
5.          case 4.*:
echo “SunOS $release” breaksw
6.          case [56].*:
echo “Solaris Srelease” breaksw
7.          endsw
breaksw case HP*:
echo HP-UX breaksw
case Linux:
echo Linux breaksw
8.  endsw
(The Command Line)
% systype
```

yoJr s, *szem type:*
SunOS 4.1.2

설명

1. 변수 release에 조작체계의 판본에 대한 계렬번호인 uname-r의 출력이 대입된다.
2. switch지령이 조작체계의 이름 uname-s의 출력을 평가한다.
3. 체계형의 SunOS이면 3행의 case지령이 실행된다.
4. 변수 release의 값이 매 표식에서 평가되여 비교된다.
5. 모든 계렬판본들에 대한 표식 4가 검사된다.
6. 모든 계렬판본들에 대한 표식 5, 6이 검사된다.
7. 내부switch명령이 끝난다.
8. 바깥switch명령이 끝난다.

7. 순환

순환지령들은 같은 명령을 여러번 실행하게 한다. C쉘은 두가지 형태의 순환 foreach순환과 while순환을 지원한다. foreach순환은 파일들의 목록이나 사용자이름들의 목록과 같이 항목들의 목록에서 지령들을 한개 항목씩 실행할 필요가 있을 때 사용한다. while순환은 어떤 조건이 만족할 때까지 한개 지령을 계속 실행하려고 할 때 사용한다.

foreach순환 foreach지령다음에 한개 변수와 괄호안의 단어목록이 온다. 첫번째 순환때 목록에 있는 첫번째 단어가 변수에 대입된다. 목록은 왼쪽으로 한자리 옮김되고 순환이 진행된다. 순환본체에 있는 매 지령은 end명령이 나타날 때까지 실행된다. 조종은 순환의 꼭대기로 복귀한다. 목록에 있는 다음단어가 변수에 대입되고 foreach행다음에 있는 지령이 실행된다. end가 나타나면 조종은 foreach순환의 꼭대기로 복귀하고 단어목록에 있는 다음단어가 처리되는 식으로 계속된다. 단어목록이 빌 때 순환이 끝난다.

형식

```
foreach variable (wordlist)
commands end
```

실례 9-93

```
1. foreach person (bob Sam sue fred)
2. mail $person < letter
3. end
```

설명

1 . foreach지령다음에 변수 person과 괄호로 둘러 막힌 단어목록이 놓인다.

첫번째 순환때 변수 person의 값 bob가 대입된다. 일단 bob가 person에 대입되였으면 bob는 자리옮김(왼쪽으로)되고 sam이 그 목록의 처음에 놓인다. end명령이 나타날 때 조종은 순환의 꼭대기에서 시작되고 sam이 변수 person에 대입된다. 이 과정은 fred가 자리옮김되고 목록이 비여 순환이 끝날 때까지 계속된다.

2. 사용자 bob에 첫번째 순환때 파일 letter의 내용들이 전송된다.
3. end명령이 나타날 때 순환조종이 foreach로 복귀되고 목록에 있는 다음 요소가 변수 person에 대입된다.

실례 9-94

(The Command Line)

```
% cat maillist
tom
dick
harry
dan
```

(The Script - mailtomaillist)

```
     #!/bin/csh -£
     # This script is called mailtomaillist
1.   foreach person ('cat maillist')
2.       mail $person «EOF Hi $person,
         How are you?   I've missed you. Come on over
         to my place.
         Your pal,
                  $LOGNAME@'uname -n'
     EOF
3.   end
```

설명

1. 괄호안에서 지령바꿔넣기가 진행된다. 파일 maillist의 내용들이 단어목록으로 된다. 단어목록(tom, dick, harry, dan)에 있는 매 이름이 변수 person에 차례로 대입된다. 순환명령이 실행되고 end가 나타난 다음에 조종이 foreach로 복귀되며 한개 이름이 목록으로부터 자리옮김되여 변수 person에 대입된다. 목록에 있는 다음번 이름이 방금 자리옮김된 이름을 대신한다. 따라서 목록의 크기는 1만큼 감소한다. 이 과정이 모든 이름들이 자리옮김되여 목록이 빌 때까지 계속된다.
2. here문서가 사용된다. 첫번째 EOF로부터 마지막 EOF까지 입력이 mail프로그람으로 전송된다(마지막 EOF는 행의 왼쪽 시작점에 있어야 하며 그 주위에 공백이 있으면 안된다.). 목록에 있는 매 사람에게 mail통보가 보내진다.
3. foreach순환에 대한 end명령은 이 순환안에서 실행되는 행블로크의 끝을 표시한다. 조종은 순환의 꼭대기로 복귀한다.

실례 9-95

```
1.  foreach file (*.c)
2.          cc $file -o $file:r
    end
```

설명

1. foreach지령에 대한 단어목록은 .c로 끝나는 현재등록부에 있는 파일들의 목록이다(즉 모든 c원천파일들이다.).
2. 단어목록에 있는 매 파일이 번역된다. 실례로 처리되는 첫번째 파일이 progame.c이면 쉘은 cc지령행을 다음과 같이 전개한다.

   cc program. c-o program

   :r는 .c확장자를 없애게 한다.

실례 9-96

(The Command Line)

```
1.  % runit fl f2 £3 dir2 dir3
```

(The Script)

```
    #!/bin/csh -f
    # This script is called runit.
    # It loops through a list of files passed, as
    # arguments.
2.  foreach arg   ($*)
3.          if ( -e $arg ) then
    ...                     Program code continues here
    else
    ...                     Program code cont-inues here
    endif
4.  end
5.  echo "Program continues here"
```

설명

1. 스크립트의 이름은 runit이다. 지령행인수들은 f1, f2, f3, dir2, dir3이다.
2. $*변수는 지령행에서 넘겨 진 모든 인수들(위치파라메터들)의 목록을 표시한다. foreach지령이 단어목록에 있는 매 단어 f1, f2, f3, dir2, dir3을 차례로 처리한다. 순환할 때마다 목록에 있는 첫번째 단어가 변수 arg에 대입된다. 한개 단어가 대입된후 그것은 자리옮김(왼쪽으로)되고 다음번 단어가 arg에 대입되는데 이것은 목록이 빌 때까지 계속된다.
3. 이 블로크에 있는 지령들이 end명령이 나타날 때까지 목록에 있는 매항에 대하여 실행된다.
4. 단어목록이 빈 다음에 end명령이 순환을 끝낸다.
5. 순환이 끝난후 프로그람이 실행을 계속한다.

while순환 while순환은 식을 평가하여 그 식이 참(령이 아닌)인 동안 while지령 다음의 지령들은 end명령이 나타날 때까지 실행된다. 그다음 조종이 while식으로 되돌아 가고 식이 평가되는데 그 식이 여전히 참이면 지령들이 반복실행된다. 이런 식으로 순환이 계속된다. while식이 거짓일 때 순환은 끝나고 조종은 end명령다음부터 시작된다.

실례 9-97

(The Script)

```
      #!/bin/csh -f
1.  set num = 0
2.  while ($num < 10)
3.        echo $num
4.        ® num++          # See arithmetic.
5.  end
6.  echo "Program continues here"
```

설명

1. 변수 num이 초기값 0으로 설정된다.
2. while순환이 시작되고 식이 검사된다. num의 값이 10보다 작으면 식이 참으로 되고 행 3과 4가 실행된다.
3. num의 값이 순환될 때마다 매번 현시된다.
4. 변수 unm의 값이 증가된다. 이 명령이 생략되였으면 순환이 무한히 계속된다.
5. end명령은 실행명령들의 블로크를 끝낸다. 이 행이 나타날 때 조종은 while순환의 꼭대기로 되돌아 가고 식이 다시 검사된다. 이것이 while식이 거짓으로 될 때(즉 $num이 10일 때)까지 계속된다.
6. 순환이 끝난후 프로그람실행이 계속된다.

실례 9-98

(The script)

```
      #!/bin/csh -f
1.  echo -n   "Who wrote \ "War and Peace\ "?"
2.  set answer = $<
3.  while ("$answer" != "Tolstoy")
echo "Wrong, try again\ !"
4.        set answer = $<
5.  end
6.  echo Yeah!
```

설명

1. 사용자입력이 요구된다.
2. 변수 answer에 사용자가 입력하는것이 대입된다.

3. while지령이 식을 검사한다. 만일 $answer의 값이 문자렬 Tolstoy와 꼭같지 않으면 통보 wrong, try again!가 인쇄되고 프로그람이 사용자입력을 대기한다.
4. 변수 answer에 새로운 입력이 대입된다. 이 행이 중요하다. 만일 변수 answer의 값이 변하지 않으면 순환식은 거짓으로 되지 않는다. 따라서 순환이 무한히 계속되게 한다.
5. end명령은 while순환안에서 코드블로크를 끝낸다.
6. 사용자가 Tolstoy를 입력하면 순환식이 거짓으로 검사되고 조종이 이 행으로 넘어 간다. Yeah!가 인쇄된다.

repeat 지령 repeat 지령은 2 개의 인수 즉 한개 수와 한개 지령을 가진다. 지령은 수만큼 반복실행된다.

실례 9-99

```
% repeat 3 echo hello
hello
hello
hello
```

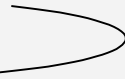
설명

echo지령이 세번 실행된다.

8. 순환지령

shift지령 shift지령은 배렬이름을 인수로 가지지 않으면 argv배렬을 왼쪽으로부터 한개 단어만큼 자리옮김하여 argv배렬의 크기를 하나 감소시킨다. 일단 배렬의 요소가 자리옮김되면 없어 진다.

실례 9-100

```
(The Script)
      #!bin/csh -f
      # Script is called loop.arg-s
1.  while ($#argv)
2.      echo $argv
3.      shift
4.  end
(The Command Line)
5.  % loop.args abode
    a b c d e
    b c d e
    c d e
    d e
    e
```

설명

1. $#argv는 지령행인수들의 수와 같다. 만일 5개의 지령행인수 a, b, c, d, e가 있다면 $#argv의 값은 첫번째 순환때 5이다. 식이 검사되고 그 결과는 5 즉 참이다.
2. 지령행인수들이 인쇄된다.
3. argv배렬이 왼쪽으로 하나 자리옮김된다. b로 시작되여 4개의 인수들만 남는다.
4. 순환끝이 나타나면 조종은 순환의 꼭대기로 다시 넘어 간다. 식이 다시 검사된다. 이때 $#argv는 4이다. 인수들이 인쇄되고 배렬이 다시 자리옮김된다. 모든 인수들이 자리옮김될 때까지 이것이 계속된다. 이때 식이 검사되면 그 값은 0 즉 실패하여 순환이 멎는다.
5. 인수 a, b, c, d, e가 argv배렬을 거쳐 스크립트에 넘겨 진다.

break지령 breck지령은 조종이 end명령다음에 시작되도록 순환에서 벗어 나기위해 사용된다. 그것은 제일 안쪽 순환에서 벗어 난다. 순환의 end명령다음에 실행이 계속된다.

실례 9-101

```
   #!/bin/csh -f
   # This script is called baseball
1. echo -n "What baseball hero died in August 1995-:
2. set answer = $<
3. while ("$answer" !- [Mm]*)
4.         echo "Wrong\ ' Try again."
           set answer = $<
5.         if ( "$answer" = ~ [Mm]* ) break
6. end
7. echo "You are a scholar."
```

설명

1. 사용자입력이 요구된다.
2. 사용자로부터의 입력이 변수 answer에 대입된다(answer:Mickey Mantle).
3. while식은 다음과 같다. answer의 값이 M이나 m으로 시작되지 않고 그다음에 임의의 수의 문자가 놓이면 순환을 시작한다.
4. 사용자가 다시 시도한다. 변수가 재설정된다.
5. 변수 answer가 M이나 m과 일치되면 순환에서 탈퇴한다. end명령으로 가서 7행에 있는 명령들을 실행한다.
6. end명령은 순환다음의 이 명령블로크를 끝낸다.
7. 순환이 끝난후 여기서 조종이 시작되고 이 행이 실행된다.

실례 9-102

```
#i/bin/csh -f
```

```
# This script is called database
1.  while (1)
            echo "Select a menu item"
2.          cat « EOF
            1) Append
            2) Delete
            3) Update
            4) Exit
    EOF
3.          set choice = $<
4.          switch ($choice)
            case 1:
                        echo "Appending"
5.                      break            # Break out of loop; not a breaksw
            case 2:
                        echo "Deleting"
                        break
            case 3 :
                        echo "Updating"
                        break
            case 4
                        exit 0
            default:
6.                      echo "Invalid choice   Try again
            endsw
7.  end
8.  echo   Program continues here'
```

설명

1. 이것을 무한순환이라고 한다. 식은 언제나 1 즉 참으로 된다.
2. 이것은 here문서이다. 차림표가 현시장치에 인쇄된다.
3. 사용자가 차림표항목을 선택한다.
4. switch지령이 변수를 검사한다.
5. 만일 사용자가 1과 4사이의 유효값을 선택하면 일치된 case표식다음의 지령이 실행된다. break명령은 프로그람이 순환에서 벗어 나 8행에서 실행을 시작하도록 한다. 이것을 endsw에서 switch를 탈퇴하는 breaksw명령과 혼돈하지 말아야 한다.
6. 기정case표식이 일치되면 즉 어느 한 추가선택도 일치되지 않으면 프로그람조종은 순환의 끝으로 넘어 가 while의 짝대기에서 다시 시작된다. while다음의 식이 언제나 참이기때문에 순환이 진행되고 차림표가 다시 현시된다.
7. while순환명령의 끝이다.
8. 순환이 끝난후 이 행이 실행된다.

겹쌓인 순환과 repeat지령 goto보다도 repeat지령을 사용하여 겹쌓인 순환에서 벗어 날수 있다. repeat지령은 이것을 continue지령과 다른 방법으로 수행한다.

실례 9-103

(Simple Script)

```
    #!/bin/csh -f -
1.  while (1)
          echo   "Hello, in 1st loop"
2.    while (1)
              echo "In 2nd loop"
            while (1)
                echo "In 3rd loop"
                repeat 3    break
            end
      end
    end
5.  echo "Out of all loops "
```

(The Output)

Hello, in 1st loop

In 2nd loop

In 3rd loop

out of all loops

설명

1. 첫번째 while순환을 시작한다.
2. 두번째 겹while순환에 들어 간다.
3. 세번째 겹while순환에 들어 간다.
4. repeat지령은 berak가 3번 실행되게 한다. 처음에 제일 안쪽 순환에서 벗어 나고 그다음 두번째 순환에서, 마지막으로 첫번째 순환에서 벗어 난다. 조종은 행 5에서 계속된다.
5. 프로그람조종이 순환이 끝난후 시작된다.

continue지령 continue명령은 제일 안쪽 순환의 꼭대기에서 실행을 시작한다.

실례 9-104

설명

1. 변수 done에 0이 대입된다.
2. 식이 검사된다. 그것은 while(!0)과 같다. 0의 부정은 참으로 평가된다(론리부정).
3. 사용자가 No, no 또는 nope(N이나 n으로 시작하는 임의의 문자렬)을 입력했으면 식은 참으로 되고 continue명령은 식이 다시 검사되는 순환의 꼭대기로 조종을 되돌린다.
4. 만일 answer가 N이나 n으로 시작되지 않으면 변수 done이 1로 재설정된다. 순환의 끝이 나타날 때 조종은 순환의 꼭대기에서 시작되고 식이 검사된다. 그것은 다음과 같다. while(!1)1의 부정은 거짓이다. 순환이 끝난다. 이것은 while순환의 끝을 표시한다.

실례 9-105

```
(The Script)
    #!/bin/csh -f
1.   if ( i -e memo ) then
         echo "memo file non existent"
          exit 1
     endif
2.   foreach person (anish bob don karl jaye)
3.         if ("$person" =~ [Kk]arl) continue
4.         mail -s "Party time" $person < memo
      end
```

설명

1. 파일시험이 진행된다. 만일 memo파일이 존재하지 않으면 사용자에게 오유통보가 전송되고 프로그람이 상태 1로 탈퇴한다.
2. 순환은 목록에 있는 매 사람을 변수 person에 차례로 대입하고 그다음에 목록에서 이름을 자리옮김하고 다음의것을 처리한다.
3. 만일 사람의 이름이 Karl 혹은 karl이면 continue명령은 foreach순환의 꼭대기에서 실행을 시작한다(Karl의 이름이 person에 대입된후 자리옮김되였기때문에 Karl에 memo가 보내지지 않는다.). 목록에 있는 다음이름이 person에 대입된다.
4. 우편목록에 있는 모두에게 karl을 제외한 memo가 보내진다.

9. 새치기처리

스크립트는 새치기건으로 새치기되면 끝나고 조종이 C쉘에로 되돌아 간다. 즉 재촉문이 다시 나타난다. onintr지령은 스크립트안에서 새치기를 처리하는데 사용된다.

그것은 탈퇴하기전에 새치기(^C)를 무시하거나 프로그람의 다른 부분으로 조종을 넘길수 있게 한다. 보통 interrupt지령은 탈퇴하기전에 《정돈》하기 위한 표식과 함께 표시된다. 인수가 없는 onintr지령은 기정동작을 복귀시킨다.

실례 9-106

(The Script)

```
1. onintr   finish
2.        < Script continues here >
3. finish:
4. onintr -              # Disable further interrupts
5. echo Cleaning temp files
6. rm $$tmp* ; exit 1
```

설명

1. onintr지령다음에 표식이름이 온다. 표식 finish는 사용자정의된 표식이다. 만일 새치기가 일어 나면 조종이 finish표식으로 넘어 간다. 보통 이 행은 스크립트의 시작에 있다. 그것은 스크립트에서 실행될 때까지 아무런 영향도 주지 않는다.
2. 프로그람이 실행될 때 ^C(새치기건)가 눌리우지 않으면 스크립트행들의 나머지가 실행된다. 새치기건이 눌리워 지면 조종은 표식으로 넘어 간다.
3. 이것은 표식이다. 새치기가 들어 올 때 프로그람은 표식다음의 명령들을 계속 실행시킨다.
4. 스크립트의 이 부분을 새치기로부터 보호하기 위하여 onintr-가 사용된다. 만일 Control-C가 입력되면 무시된다.
5. 이 행이 현시장치에 출력된다.
6. 모든 tmp파일이 지워 진다. tmp파일의 앞에 쉘의 PID($$)번호가 붙고 뒤에는 임의의 문자들이 놓인다. 프로그람은 상태를 1로 하고 탈퇴한다.

10. setuid스크립트

setuid프로그람을 림시적으로 실행시키는 사람들은 누구나 다(Setuid프로그람을 실행시키고 있는 한) 그 프로그람의 소유자로 되며 소유자와 같은 허가를 가진다. passwd프로그람은 setuid프로그람에 대한 훌륭한 실례이다.

사용자는 암호를 변화시킬 때 림시로 passwd프로그람의 실행기간만 root로 된다. 따라서 정기사용자가 보통 사용할수 없는 /etc/passwd(또는 /etc/shadow)파일에서 암호를 변화시킬수 있다. 쉘프로그람은 setuid프로그람으로 작성될수 있다.

로임이나 개인정보와 같이 정기사용자들이 보지 말아야 할 정보를 포함하고 있는 파일을 호출하는 스크립트가 있으면 쉘프로그람을 setuid프로그람으로 작성하려고 할 수 있다. 만일 스크립트가 setuid스크립트이면 그 스크립트를 실행하는 사람은 자료를 호출할수 있지만 다른 사람들은 호출하지 못한다.

setuid프로그람은 다음의 단계를 요구한다.

1. 스크립트에서 첫번째 행은 다음과 같다.

```
#!/bin/csh –feb
```

The -feb options:

-f fast start up; don't execute .cshrc
-e abort immediately if interrupted
-b this is a setuid script

2. 다음스크립트가 setuid프로그람으로 실행되도록 스크립트에 대한 허가를 변화시킨다.

% chmod 4775 script_name
or
% chmod +srx script_name
% ls –1
-rwsr-xr-x 2 ellie 512 Oct 10 17:18 script_name

11. 스크립트기억

스크립트들을 성과적으로 작성한후 보통 그것들을 공동스크립트등록부에 모아 놓고 경로를 변화시켜 스크립트들이 임의의 위치로부터 실행될수 있도록 한다.

실례 9-107

1. % **mkdir ~/bin**
2. % **mv myscript ~/bin**
3. % **vi .login**

   In .login reset the path to add ~/bin.
4. **set path = (/usr/ucb /usr /usr/etc ~/bin .)**
5. (At command line)
   % **source .login**

설명

1. 홈등록부안에 bin이라고 하는 등록부를 만든다. 등록부이름은 임의로 선택해도 된다.
2. 임의의 오유가 없는 스크립트들을 빈 등록부에 넣는다. 오유가 많은 스크립트들을 여기에 넣으면 문제들이 생긴다.
3. .login파일에로 가서 경로를 재설정한다.
4. 새로운 경로에 등록부~/bin이 있는데 여기서 쉘이 실행프로그람들을 찾는다. 그것이 경로의 끝가까이에 있으므로 스크립트들중 어느 하나와 이름이 같은 체계프로그람이 있으면 그것이 먼저 실행된다.
5. .login을 원천화하여 경로변경에 영향을 준다. 등록탈퇴하였다가 다시 가입하지 않아도 된다.

12. 내부지령

내부지령들은 UNIX지령들과 같이 디스크우에 있지 않고 C쉘의 내부코드의 부분이며 쉘내부로부터 실행된다. 만일 내부지령이 파이프행의 마지막지령이 아닌 임의의 요소로 된다면 보조쉘에서 실행된다. 내부지령들의 목록을 표 9-16에 주었다.

표 9-16. 내부지령과 그 의미

내부지령	의 미
:	null지령을 해석하지만 동작을 진행하지 않는다.
alias	지령에 대한 별명이다.
bg[% job]	배경에서 현재 또는 지정된 일감들을 실행한다.
break	제일 안쪽 foreach나 while순환에서 벗어 난다.
breaksw	switch에서 벗어 나 endsw다음에서 다시 시작한다.
case label:	switch명령에서 표식이다.
cd[dir]	쉘의 작업등록부를 dir로 변화시킨다. 인수가 주어 진다.
cddir[dir]	사용자의 홈등록부로 변화시킨다.
continue	제일 가까운 정지된 while이나 foreach의 실행을 계속한다.
default	switch명령에서 기정case표식을 현시한다. 기정표식은 모든 case표식다음에 놓여야 한다.
dir[-l]	등록부묶음을 인쇄하는데 제일 최근의것을 왼쪽에 놓는다. 현시되는 첫번째 등록부는 현재등록부이다. -l인수를 사용하면 간략화하지 않고 인쇄한다. ~표기를 사용하지 않는다.
echo[-n]list	목록에 있는 단어들을 쉘의 표준출력에 쓴다. 단어들은 공백문자에 의해 분리된다. 출력은 -n추가선택이 사용되지 않으면 행바꾸기로 끝난다.
eval지령	지령을 쉘에 대한 표준입력으로 실행하고 그 결과 생기는 지령을 실행한다. 지령이나 변수바꿔넣기전에 해석이 진행되므로 이 대입들의 결과로 생기는 지령들을 실행하는데 이것을 사용한다(실례 eval 'test-s 추가선택').
exec지령	현재쉘에서 지령을 실행시켜 그것을 끝낸다.
fg[% job]	현재 또는 지적된 일감을 전경에로 넘긴다.
Foreach var(wordlist)	379페지의 《foreach순환》을 참고
glob wordlist	단어목록에서 파일이름확장을 한다. cho와 비슷하지만 거꿀빗선(\ )이 인식되지 않는다. 단어들은 출력에서 빈값문자들에 의해 구분된다.
gotolabel	372페지에서 《goto》를 참고
Hashstat	내부하쉬표가 지령들을 찾아 내는데서(exec를 피하는데서)얼마나 효과적인가를 보여 주는 통계행을 인쇄한다. Exec는 하쉬함수가 가능한 성공을 표시하는 경로의 매 요소에 대하여 시도되며 거꿀빗선으로 시작하지 않는 매 요소에서 실행된다.
history[-hr][n]	리력목록을 현시한다. n이 주어 지면 n개의 가장 최근의 사건들만 현시한다.
-r	인쇄순서를 제일 오래 된것을 첫번째로 하지 않고 제일 최근의것이 첫번째로 되게 절환한다.

(표계속)

내부지령	의 미
-h	안내수들이 없이 리력목록을 현시한다. 이것은 -h추가선택을 사용하여 원천화하는데 적합한 파일을 만들기 위해 쓰인다.
If(expr)	364페지의 《흐름조종과 조건구성체》를 참고
else if(expr2) then	364페지의 《흐름조종과 조건구성체》를 참고
Jobs[-l]	일감조종을 받고 있는 일감들을 현시한다.
-l	표준정보외에 ID들을 현시한다.
kill[-sig][pid][%job] killH	기정으로 혹은 지정된 신호로써 TERM(끝)신호를 지정된 ID나 표시된 일감 또는 현재일감으로 보낸다. 신호들은 번호나 이름에 의해 주어 진다. 기정값이 없다. kill을 입력하면 신호를 현재일감에로 보내지 않는다. 보내진 신호가 TERM(끝)이나 HUP(정지)이면 일감이나 프로쎄스에 COMT(계속)신호도 보내진다.
-l	전송될수 있는 신호이름들을 현시한다.
limit[-h][resource (max-use)]	현재프로쎄스나 그것이 파생시킨 임의의 프로쎄스에 의한 사용을 제한하여 지정된 자원에서 max-use를 초과하지 않게 한다. 만일 max-use가 생략되면 현재한계를 인쇄한다. 자원이 생략되면 모든 한계들을 인쇄한다.
-h	현재한계대신 고정한계를 사용한다. 고정한계는 현재한계의 값우에 추가된다. 주사용자만이 고정한계를 올릴수 있다. 자원은 CPU의 최대프로쎄스당 시간(초)인 cputime, 프로쎄스에 대한 최대자료크기(탄창포함)인 datasize, 허용되는 제일 큰 단일파일인 filesize, 프로쎄스에 대한 최대탄창크기인 stacksize, 기본찍어내기의 최대크기인 coredump와 파일서술자에 대한 최대값인 descriptors이다.
login[username|-p]	login쉘을 끝내고 login(1)을 호출한다. .logout파일이 처리되지 않는다. 사용자이름이 생략되면 login은 사용자의 이름을 재촉한다.
-p	현재환경(변수들)을 보존한다.
logout	login쉘을 끝낸다.
nice[+n|-n][command]	쉘이나 지령에 대한 프로쎄스우선권값을 n만큼 증가시킨다. 우선권값이 클수록 프로쎄스의 우선권은 작아 지고 더 천천히 실행된다. command가 생략되면 nice가 현재쉘에 대한 값을 증가시킨다. 증가가 지적되지 않으면 nice는 nice값을 4로 설정한다. nice값의 범위는 -20부터 19사이이다. 이 범위밖에 있는 n의 값은 값을 더 낮거나 더 높은 경계에로 각각 설정한다.
+n	프로쎄스우선권값을 n만큼 증가시킨다.
-n	n만큼 감소시킨다. 이 인수는 주사용자에 의해서만 사용될수 있다.
nohup[command]	HUP(정지)신호를 무시하고 지령을 실행시킨다. 인수가 없으면 스크립트의 나머지부분에서 HUP를 무시한다.
notify[%job]	현재 또는 지정된 일감의 상태가 변할 때 사용자에게 비동시적으로 알려 준다.

(표계속)

내부지령	의 미
onintr[-|label]	새치기상태에서 쉘의 동작을 조종한다. 인수가 없으면 onintr는 새치기상태에서 쉘의 기정동작을 복귀한다(쉘은 쉘스크립트들을 끝내고 말단지령입력준위로 되돌아 간다.). 덜기부호인수가 있으면 쉘은 모든 새치기를 무시한다. label인수가 있으면 쉘은 새치기가 접수되거나 자식프로쎄스가 새치기되여 끝날 때 goto label을 실행한다.
popd[+n]	등록부탄창을 뽑아서 새로운 꼭대기등록부에로 등록부를 변화시킨다. 등록부탄창들의 요소들이 꼭대기에서 시작하여 0부터 번호가 매겨 진다.
+n	탄창에 있는 n번째 입력을 버린다.
pushd[+n|dir]	등록부를 등록부탄창에 넣는다. 인수가 없으면 꼭대기 2개 요소들을 바꾼다.
+n	n번째 입력을 탄창의 꼭대기에 교체하고 그것으로 등록부를 변경한다.
dir	현재작업등록부를 탄창에 넣고 dir로 변화시킨다.
rehash	추가된 새로운 지령들을 고려하여 path변수에 표시된 등록부들의 내용들로 내부하쉬표를 다시 계산한다.
repeat count command	지령을 count만큼 반복한다.
set[var[=value]]	337페지의 《변수》를 참고
setenv[VAR[ward]]	337페지의《변수》를 참고한다. 가장 일반적으로 사용되는 환경변수들인 USER, TERM과 PATH는 자동적으로 csh변수들인 user, term과 path로 넘어가거나 그것들로부터 넘겨 진다. 이것들을 위해서 setenv를 사용할 필요는 없다. 또한 쉘은 csh변수 cwd가 변할 때마다 그로부터 환경변수 pwd를 설정한다.
shift[variable]	argv의 요소들 즉 variable이 주어 지면 왼쪽으로 자리옮김되고 첫번째 요소가 없어 진다. variable이 설정되지 않던가 빈값을 가지면 오유로 된다.
source[-h]nameh	name으로부터 지령을 읽는다. source지령들은 겹싸여 질수 있지만 그것들이 너무 깊게 겹싸지면 쉘은 파일서술자에서 벗어 날수 있다. 원천화된 파일에 있는오유는 임의의 준위에서 모든 겹싸여 진 source 지령들을 끝낸다. 이것은 보통 변수설정들이 현재쉘에서 처리되도록 하기 위해 .login이나 .cshrc파일을 재실행시키는데 사용된다. 즉 쉘은 자식쉘을 작성하지 않는다. 리력목록에 있는 파일이름으로부터 지령들을 실행하지 않고 리력목록에 놓는다.
stop[%iob]	현재 또는 지정된 배경일감을 정지한다.
suspend	쉘에 ^Z로 정지신호가 전송된것처럼 쉘을 그 경로에서 정지시킨다. su에 의해 파생된 쉘들을 정지시킬 때 이것을 많이 쓴다. ...
switch(string)	376페지의 《switch지령》을 참고
time[command]	인수가 없으면 C쉘과 그의 자식쉘들에 의해 사용된 총 시간을 인쇄한다. 지령을 선택하면 그 지령을 실행하고 그것이 사용한 총 시간을 인쇄한다.

(표계속)

내부지령	의 미
umask[value]	파일작성마스크를 현시한다. 값이 있으면 파일작성마스크를 설정한다. 8진수로 주어 지는 값은 파일에 대하여 666, 등록부에 대해서는 777의 허가와 안맞음론리합되여 새로운 파일에 대한 허가를 준다. 허가는 umask를 통해 추가할수 없다.
unalias pattern	패턴과 일치되는 별명들을 버린다. unalias*에 의해 모든 별명들이 없어 진다.
Unhash	내부하쉬표를 쓸수 없게 한다.
unlimit[-h][resource]	자원에 대한 한계를 없앤다. 그 어떤 자원도 지정되지 않으면 모든 자원한계들이 없어 진다. 자원한계이름목록에 대하여 limit지령을 참고한다.
-h	대응하는 하드한계를 지우기한다. 주사용자만이 이것을 할수 있다.
unset pattern	그 이름들이 패턴과 일치되는 변수들을 지운다. 'unset*'에 의해 모든 변수들이 지워 진다. 이렇게 하면 부정적영향이 나타날수 있다.
unsetenv variable	변수들을 환경으로부터 지우기한다. unset를 사용한 때와 같이 패턴일치이 진행되지 않는다.
Wait	재촉문이 나타나기전에 배경과제가 끝나기를(또는 새치기)기다린다.
while(expr)	382페지에서 《while순환》을 참고
%[job][&]	현재 또는 지정된 과제를 전경으로 넘긴다. &기호가 있으면 배경에서 과제를 계속 실행한다.
@ [var=expr] @ [varcn]=expr]	인수가 없으면 모든 쉘변수들에 대한 값들을 현시한다. 인수가 있으면 변수 var 즉 var의 값에 있는 n번째 단어 expr가 현시하는 값으로 설정된다.

C쉘련습

련습문제 19: 기동

1. init프로쎄스는 무엇을 수행하는가?
2. login프로쎄스의 기능은 무엇인가?
3. 어떤 쉘을 사용하고 있는가를 어떻게 아는가?
4. 등록가입쉘을 어떻게 변화시키는가?
5. .cshrc파일과 .login파일의 차이를 설명하시오. 어느것이 먼저 실행되는가?
6. .cshrc파일을 다음과 같이 편집하시오.
   ㄱ. 자체로 3개의 별명을 만드시오.
   ㄴ. 재촉문을 재설정하시오.

다음의 변수들을 설정하고 매 변수뒤에 그것이 무엇을 하는가를 설명하는 해설문을 놓으시오.

noclobber *# Protects clobbering files from redirection overwriting*

history

ignoreeof

savehist

filec

7. 다음의것을 입력하시오.
   source.cshrc
   source지령은 무엇을 하는가?
8. .login파일을 다음과 같이 편집하시오.
   ㄱ. 사용자를 환영하시오.
   ㄴ. 홈등록부가 경로에 없다면 거기에 추가하시오.
   ㄷ. .login파일을 원천화하시오
9. history를 입력하시오. 무엇이 출력되는가?
   ㄱ. 마지막지령을 어떻게 재실행시키는가?
   ㄴ. echo abc를 입력하시오.

   history지령을 사용하여 마지막인수 c만을 가지고 echo지령을 재실행하시오.

련습문제 20: 쉘메타문자

1. 재촉문에서 다음의것을 입력하시오.
   touch ab abc a2 a2 a3 a11 a12 ba ba.1 ba.2 filex filey Abc ABc ABc2 abc
2. 다음의 기능을 수행하는 지령들을 쓰고 검사하시오.
   ㄱ. a로 시작하는 모든 파일들을 현시하시오.
   ㄴ. 적어도 한개 수자로 끝나는 모든 파일들을 현시하시오.
   ㄷ. a나 A로 시작하는 모든 파일들은 현시하시오.
   ㄹ. 점과 그뒤에 한개 수자가 오는 모든 파일들을 현시하시오.
   ㅁ. 문자 a가 2개 있는 모든 파일들을 현시하시오.
   ㅂ. 모든 문자들이 대문자이고 3개 문자로 된 파일들을 현시하시오.
   ㅅ. 11 또는 12로 끝나는 파일들을 현시하시오.
   ㅇ. x나 y로 끝나는 파일들을 현시하시오.
   ㅈ. 한개 수자, 대문자 또는 소문자로 된 파일들을 현시하시오.
   ㅊ. 한개의 b를 가지는 모든 파일을 현시하시오.
   ㅋ. a로 시작되는 2개 문자 파일들을 지우시오.

련습문제 21: 방향바꾸기

1. 말단과 련관된 3개의 파일흐름의 이름들은 무엇인가?
2. 파일서술자는 무엇인가?
3. 다음의것을 하기 위해 어떤 지령을 사용하는가?
   ㄱ. ls지령의 출력을 lsfile이라고 하는 파일에로 방향바꾸기하려면?
   ㄴ.date지령의 출력을 lsfile에로 방향바꾸기하고 추가하려면?
   ㄷ. who지령이 출력을 lsfile에로 방향바꾸기하려면? 무슨 일이 생기는가?
4. cp만을 단독으로 입력할 때 어떤 일이 생기는가?
   ㄱ. 우의 실례로부터 생기는 오유통보를 파일에 어떻게 기억하는가?
5. find지령을 사용하여 현재등록부로부터 시작되고 등록부형인 모든 파일들을 찾으시오.
   표준출력은 found라고 하는 파일에 기억하고 오유들은 found.errs라고 하는 파일에 기억하시오.
6. noclobber는 무엇인가? 그것을 어떻게 무시하는가?
7. 3개 지령의 출력을 취하여 gotemail라고 하는 파일에로 방향바꾸기하시오.
8. ps지령, wc지령과 함께 파이프를 사용하여 현재 몇개의 프로쎄스들을 실행하고 있는가를 알아 보시오.

련습문제 22: 첫번째 스크립트

1. 다음의것을 진행하는 greetme라고 하는 스크립트를 작성하시오.
   ㄱ. 사용자를 맞이하시오.
   ㄴ. 자료와 시간을 인쇄하시오.
   ㄷ. 이 달에 대한 력서를 인쇄하시오.
   ㄹ. 콤퓨터이름을 인쇄하시오.
   ㅁ. 어미등록부에 있는 모든 파일들의 목록을 인쇄하시오.
   ㅂ. 지금 실행하고 있는 모든 프로쎄스들을 인쇄하시오.
   ㅅ. TERM, PATH, HOME변수들의 값을 인쇄하시오.
   ㅇ. Please couldn't you loan me $ 50.00?를 인쇄하시오.
   ㅈ. 사용자에게 인사를 하고 현재시간을 알려 주시오(date지령을 위해 사용지도서페지를 보시오.).
2. 스크립트를 실행형으로 만드시오.

   ```
   chmod + x greetme
   ```
3. 스크립트의 첫번째 행은 무엇이였는가?

련습문제 23: 사용자입력얻기

1. 다음의것을 하는 nosy라고 하는 스크립트를 작성하시오.
   ㄱ. 사용자의 완전한 이름 즉 첫번째, 마지막, 가운데 이름을 물어 보시오.
   ㄴ. 사용자를 그의 첫번째 이름으로 맞이하시오.
   ㄷ. 사용자의 출생년도를 물어 보고 그 나이를 계산하시오.
   ㄹ. 사용자의 등록가입이름을 물어 보고 그의 ID를 인쇄하시오(/etc/passwd로부터).
   ㅁ. 사용자에게 그의 홈등록부를 알려 주시오.
   ㅂ. 사용자에게 그가 실행하고 있는 프로쎄스를 보여 주시오.
   ㅅ. 사용자에게 요일과 현재시간을 알려 주시오.
   출력은 다음과 같은 형식으로 되여야 한다.
   The day of the week is Tuesday and the current time is 04:07:39PM
2. datefile이라고 하는 본문파일을 작성하시오(이 파일이 이미 제공되여 있지 않으면). 매 입력은 옹근두점에 의해 분리되는 마당들로 이루어 진다. 마당들은 다음과 같다.
   ㄱ. 성과 이름
   ㄴ. 전화번호
   ㄷ. 주소
   ㄹ. 생년월일
   ㅁ. 로임
3. 다음의것을 수행하는 lookup이라는 스크립트를 작성하시오.
   ㄱ. 스크립트이름, 자기이름, 자료 그리고 이 스크립트를 작성하는 목적 등으로 설명문을 작성하시오. 이 스크립트를 작성하는 목적은 datafile을 분류된 순서로 현시하는것이다.
   ㄴ. datafile을 마지막이름으로 분류하시오.
   ㄷ. 사용자에게 datafile의 내용들을 보여 주시오.
   ㄹ. 사용자에게 파일에 있는 입력들의 수를 알려 주시오.
4. 스크립트를 오유수정하기 위해 echo지령과 verbose지령을 사용하시오. 이 지령들을 어떻게 사용했는가?

련습문제 24: 지령행인수

1. 다음의것을 수행하는 rename이라고 하는 스크립트를 작성하시오.
   ㄱ. 2개의 파일이름을 지령행인수로 하시오. 첫번째 파일은 낡은 파일이고 두번째 파일은 새것이다.
   ㄴ. 낡은 파일이름을 새로운 파일이름으로 고치시오.

ㄷ. 등록부의 파일들을 현시하여 변화를 보여 주시오.

2. 다음의것을 수행하는 cheeking이라고 하는 스크립트를 작성하시오.

ㄱ. 지령행인수인 사용자등록가입이름을 취하시오.

ㄴ. 지령행인수가 제공되였는가를 검사해 보시오.

ㄷ. 사용자가 /etc/passwd파일에 있는가를 검사한다. 만일 그렇다면 다음의것을 인쇄하시오.

Found <user>in the/etcl/passwd file

그렇지 않으면 다음의것을 인쇄하시오.

No such user on our system.

련습문제 25: 조건명령과 파일시험

1. lookup스크립트에서 사용자가 datafile에 입력을 추가하려고 하는가를 물어 보시오. 만일 yes 또는 y이면

ㄱ. 사용자에게 새로운 이름, 전화번호, 주소, 생년월일, 로임을 물어 보시오. 매 항목은 다른 변수에 기억된다. 마당들사이에 옹근두점을 주며 정보를 datafile에 추가한다.

ㄴ. 파일을 마지막이름으로 분류하시오. 사용자에게 입력을 추가했다는것을 알려 주고 번호가 앞에 붙은 행을 현시한다.

2. checking을 다시 작성하시오.

ㄱ. 이름 지어 진 사용자가 /etc/passwd파일에 있는가를 검사한후 프로그람은 사용자가 등록가입되는가를 검사한다. 만일 그렇다면 프로그람은 실행되고 있는 모든 프로쎄스들을 인쇄한다. 그렇지 않으면 사용자에게 다음의것을 알려 준다.

<user> is not logged on"

3. lookup스크립트가 실행되는가는 datafile에 달려 있다. lookup스크립트에서 datafile이 존재하던 그것이 읽고 쓰기할수 있는것인가를 검사한다.

4. lookup스크립트에 다음과 같은 차림표를 추가한다.

[1] 입력추가

[2] 입력지우기

[3] 입력보기

[4] 탈퇴

5. 이미 입력추가를 스크립트에서 작성했다. 입력추가프로그람은 이름이 datafile에 이미 있는가를 검사하고 있으면 사용자에게 그것을 알려 주는 코드를 포함하고 있어야 한다. 만일 이름이 없으면 새로운 입력을 추가한다.

6. 입력지우기, 입력보기, 탈퇴기능에 대한 코드를 작성하시오.

7. 스크립트지우기부분은 입력을 지우기전에 그것이 존재하는가를 검사해야 한다. 입력이 존재하지 않으면 사용자에게 통지한다. 그렇지 않으면 입력

을 지우고 사용자에게 그것을 지웠다는것을 알려 준다. 탈퇴할 때 반드시 수자를 사용하여 적당한 탈퇴상태를 표시하시오.

8. 지령행으로부터 탈퇴상태를 어떻게 검사하는가?

련습문제 26: Switch명령

1. Switch명령을 사용하여 다음의 스크립트를 다시 작성하시오.

```
#!/bin/csh –f
# Grades program
echo –n "What was your grade on the test? "
set score = $<
if ( $grade >= 90 && $grade <=100 ) then
echo you got an A\ !
else if ( $grade >= 80 && $grade < 89 ) then
echo you got a B.
else if ( $grade >= 79 && $grade < 79 ) then
echo "You 're average."
else if ( $grade >= 69 && $grade < 69 ) then
echo Better study harder
else
echo Better luck next time.
endif
```

2. 매 차림표항목에 대해 Switch명령을 사용하여 lookup스크립트를 다시 작성하시오.

련습문제 27: 순환

1. 사용자들의 목록에서 한번에 한명에게 들놀이초청장을 보내는 picnic라고 하는 프로그람을 작성하시오. 사용자들의 목록은 friends라는 파일에 있다. friends파일에 있는 사용자들중 한명은 popeye이다.
   ㄱ. 초청장은 invite라는 다른 파일에 있다.
   ㄴ. 파일시험을 사용하여 2개의 파일이 다 존재하며 읽을수 있는가를 검사하시오.
   ㄷ. 사용자들의 목록을 반복하기 위해 순환이 사용된다. popeye가 나타날 때 그를 뛰여 넘고(즉 그는 초청장을 받지 못한다.) 목록에 있는 다음사용자에게 초청장을 보내시오.

ㄹ. 목록을 초청장을 받는 매 사람의 이름들과 함께 기억하시오. 배렬을 작성하여 하시오. 목록에 있는 매 사람에게 우편이 보내진후 그 우편을 받은 사람들의 수와 그 이름들의 목록을 인쇄하시오.

* 시간이 있으면 매 사용자가 자기의 이름이 있는 편지를 받을수 있도록 invite파일을 수정할수 있다. 실례로 다음과 같이 통보를 쓸수 있다.

DearJohn
I hope you can make it to our piunic

이렇게 하려면 invite파일을 다음과 같이 쓸수 있다.

Dear XXX
I hope you can make it to our pichic

set나 awk를 사용하여 XXX를 사용자이름으로 치환할수 있다(사용자이름이 언제나 소문자이므로 사용자이름에 대문자를 쓰는것이 힘들수 있다.).

2. 다음과 같은 새로운 차림표항목을 lookup스크립트에 추가하시오.

[1] 입력추가
[2] 입력지우기
[3] 입력변경
[4] 입력보기
[5] 탈퇴

사용자가 유표입력을 선택한 다음 그 기능이 끝났을 때 사용자에게 차림표를 다시 보려하는가를 물어 본다. 만일 무효입력이 입력되면 프로그람은 다음의것을 인쇄한다.

Invalid entry try again.
차림표가 다시 현시된다.

3. lookup스크립트에서 입력보기아래에 부분차림표를 만드시오. 사용자에게 선택된 인물들에 대한 특수정보를 보려고 하는가를 물어 본다.

ㄱ. 전화번호
ㄴ. 주소
ㄷ. 생년월일
ㄹ. 로임

4. 표식을 사용하여 스크립트에 onintr지령을 추가한다. 프로그람이 표식에서 실행을 시작할 때 임의의 림시파일들은 지워 지고 사용자에게 인사하며 프로그람은 탈퇴한다.

제 10 장. Korn쉘

제 1 절. 대화형Korn쉘

일련의 프로세스들이 먼저 실행된 다음에 Korn쉘은 재촉문을 현시한다. 그림 10-1에 이 과정에 대하여 보여 주었다.

1. 기동

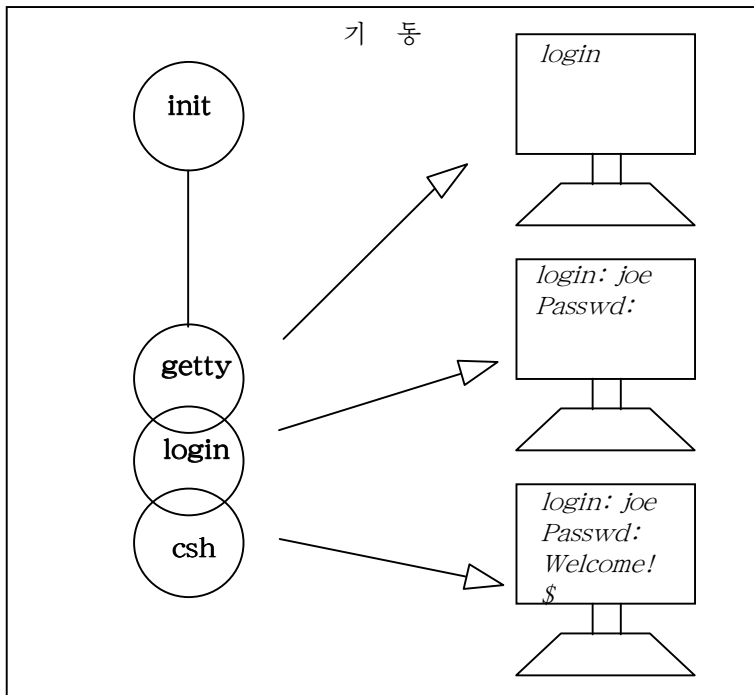



그림 10-1. 체계기동과 Korn 쉘

첫번째로 실행되는 프로세스를 init, PID 1이라고 한다. 이 프로세스는 inittab(System V)라고 하는 파일로부터 지령을 받거나 getty(BSD)프로세스를 파생시킨다. 이 프로세스들은 말단포구들을 열어 입력을 받아 들이는 표준입력(stdin)과 표준출력(stdoudt) 그리고 표준오유(stderr)를 내보내는 곳을 제공하며 현시장치에 등록가입재촉문을 현시한다. 그다음에 /bin/login프로그람이 실행된다. login프로그람은 통과암호를 재촉하고 해득하여 검증한 다음 초기환경을 설정하여 passwd파일의 마지막입력인 등록가입쉘 /bin/kdh를 기동시킨다. ksh프로그람은 체계파일 /etc/profile을 찾아 그 지령들을 실행시킨다. 그다음에 사용자의 홈등록부에서 초기화파일 .profile과 .kshrc라고 하는 환경파일을 찾는다. 이 파일로부터 지령들을 실행하면 화페기호($)재촉문이 현시장치에 나타나며 Korn쉘이 지령들을 대기한다.

2. 환경

초기화파일 /etc/profile에 있는 지령들을 실행한 다음 사용자의 홈등록부에 있는 초기화파일이 실행된다. .profile이 실행된 다음 .kshrc파일이라고 하는 ENV파일이 실행된다.

/etc/profile파일 /etc/profile은 사용자가 등록가입하고 Korn쉘이 기동할 때 과제들을 수행하도록 체계관리자가 설정하는 전체 체계에 대한 읽기파일이다. 이것은 체계에서 Bourne쉘과 Korn쉘사용자들에게 다 유용하며 보통 새로운 우편에 대한 우편입출력완충기를 검사하고 /etc/motd파일로부터 그날의 통보를 현시하는것과 같은 과제들을 수행한다. 다음의 본문은 /etc/profile의 실례이다. /etc/profile의 매개 행에 대하여 8장의《대화형Bourne쉘》을 참고한다.

실례

```
# The profile that logins get befire using their own   .profile
  trap " " 2 3
export LOGNAME PATH   # Initially set by /bin/login
if [ "$TERM" = " " ]
then
    if /bin/i386
    then        # Set the terminal type
         TERM=AT386
    Else
         TERM=sun
    Fi
    Export TERM
Fi
# Login and –su shells get /etc/profile services.
# -rsh is given its environment in the .profile.
case "$0" in
-sh | -ksh | -jsh )
     if [ ! –f .hushlogin ]
     then
        /usr/sbin/quota
# Allow the user to break the Message-of-The-Day only.
    Trap "trap ' ' 2" 2
    /bin/cat –s /etc/motd
    #Display the message of the day
```

```
trap " " 2
/bin/mail –E
case $? In
0) # Check for new mail
echo "You have new mail."
;;
2)   echo "You have mail."
;;
esac
fi
esac
umask 022
trap 2 3
```

.profile파일 .profile파일은 사용자정의초기화파일로서(Bourne와 Korn쉘에 의해) 등록가입할 때 한번 실행되며 홈등록부에 있다. 이 파일은 사용자가 작업환경을 전용화하고 변경시킬수 있게 한다. 환경변수와 말단설정들은 보통 여기서 설정되며 windows응용프로그람이나 dbm이 초기화되였으면 여기서 기동된다. 만일 .profile파일에 ENV라고 하는 특수변수가 있으면 그 변수에 대입되여 있는 파일이름이 다음에 실행된다. ENV파일을 보통 .kshrc라고 한다. 여기에는 별명들과 set-o지령들이 있다. ENV파일은 ksh자식쉘이 파생될 때마다 실행된다. 다음파일들의 행들은 의의가 있는것이 아니지만 반출하는 변수, 리력, 탐색경로 등과 같은 모든 개념들은 이 책에서 구체적으로 설명한다.

실례 10-1

```
1.  set –o allexport
2.  term=vt102
3.  HOSTNAME
4.  HISTSIZE=50
5.  EDITOR=/usr/ucb/vi
6.  ENV=$HOME/.kshrc
7.  PATH=$HOME/bin:/usr/bin:/usr/bin:\
     /usr/local:/etc:/bin:/usr/bin:/usr/local\
     /bin:/usr/hosts:/usr/5bin:/usr/etc:/usr/bin:.
8.  PS1="$HOSTNAME ! $ "
9.  set +o allexport
10. alias openwin=/usr/openwin/bin/openwin
11. trap '$HOME/.logout' EXIT
12. clear
```

설명

1. allexport추가선택을 설정하여 작성된 모든 변수들을 자동적으로 넘겨 준다 (자식쉘에서 사용할수 있게 만들어 진다.).
2. 말단을 vt102로 설정한다.
3. 변수 HOSTNAME에 이 콤퓨터의 이름 $(uname -n)을 대입한다.
4. HISTSIZE변수에 50을 대입한다. 리력파일로부터 50개 행들이 사용자가 history를 입력할 때 말단에 현시된다.
5. EDITOR변수에 vi편집기에 대한 경로이름을 대입한다. mail과 같은 프로그람은 편집기가 어디에서 작업하겠는가를 선택할수 있게 한다.
6. ENV변수에 홈등록부($HOME)에로의 경로와 앞으로 korn쉘의 전용설정들을 포함한 파일이름을 대입한다. .profile이 실행된 다음 ENV파일이 실행된다. ENV파일의 이름은 임의로 선택할수 있다. 보통 .kshrc라고 한다.
7. 탐색경로를 정의한다. 그것은 재촉문이나 스크립트파일에서 입력된 지령들을 탐색할 때 쉘이 사용하는 등록부들의 목록을 옹근두점으로 분리한것이다. 쉘은 지령을 찾기 위해 경로의 매 요소들을 왼쪽에서부터 오른쪽으로 탐색한다. 끝에 있는 점은 현재작업등록부를 나타낸다. 만일 표시된 등록부에서 지령을 찾을수 없으면 쉘은 현재등록부에서 찾는다.
8. 기정으로 화페기호($)인 1차재촉문을 주콤퓨터의 이름과 리력파일에 있는 현재지령의 번호 그리고 화페기호($)로 설정한다.
9. allexport추가선택을 해제한다.
10. 별명은 지령에 대한 가명이다. openwin별명에 openwin지령의 완전한 경로이름이 대입되는데 openwin지령은 Sun의 windows응용프로그람을 기동한다.
11. trap지령은 이 쉘을 탈퇴할 때 즉 등록해제될 때 .logout파일을 실행한다. .logout파일은 탈퇴할 때 실행되는 그 지령들을 포함하고 있는 사용자정의된 파일이다. 실례로 등록탈퇴하거나 림시파일을 지우거나 끝낼 때의 시간을 기억시킬 필요가 있을수 있다.
12. clear지령은 현시장치를 지운다.

ENV파일 ENV변수에 대화형ksh나 ksh프로그람(스크립트)이 기동될 때마다 실행되는 파일이름이 대입된다. ENV변수는 .profile에서 설정되며 ksh의 특수변수들과 별명들이 있는 파일이름을 그 변수에 대입한다. 이름은 습관상 .kshrc라고 하지만 그것을 임의로 이름 지을수 있다(ENV파일은 우선권추가선택이 설정될 때 처리되지 않는다. 표 10-1에서 보여 준다.).

실례 10-2

```
1.  set –o trackall

2.  set –o vi

3.  alias l=’ls –laF’
    alias ls=’ls-aF’
    alias hi=’fc –1’
    alias c=clear
```

```
4.  function pushd { pwd > $HOME/.lastdir.$$ ; }
    function popd { cd $(< $HOME/.LASTDIR.$$) ;
             rm $HOME/.lastdir.$$; pwd; }
    function psg { ps –ef | egerp $1 | egrep; }
    function vg { vgrind –s11 –t $* | lpr –t ; }
```

설명

1. 탐색된 별명들에 대한 set추가선택을 설정한다(완성된 서술은 418페지의 《별명》에서 보여 준다.).
2. vi편집기에 대한 set추가선택을 리력파일의 즉시 편집을 위해 설정한다(411페지의 《지령행리력》에서 보여 준다.).
3. 지령들에 대한 별명들(가명들)이 정의된다.
4. 함수들을 이름 짓고 정의한다(451페지의 《함수》에서 보여 준다.).

Set –o추가선택 –o추가선택을 사용하면 set지령은 추가선택들을 취할수 있다. 추가선택들은 쉘환경을 전용화할수 있게 한다. 그것들을 설정 또는 해제할수 있으며 보통 ENV파일에서 설정한다.

형식

```
set –o 추가선택   추가선택들을 설정된다.
set +o 추가선택   추가선택들을 해제한다.
set –[a-z] 추가선택 간략이다.; –부호는 추가선택들을 설정한다.
set +[a-z] 추가선택 간략이다.; +부호는 추가선택들을 해제한다.
```

실례 10-3

```
Set –o allexport
set +o allexport
set –a
set +a
```

설명

1. allexport추가선택을 설정한다. 이 추가선택은 모든 변수들이 자식쉘들에 자동적으로 넘겨 지게 한다.
2. allexport추가선택을 해제한다. 모든 변수들은 현재쉘에서 국부변수로 된다.
3. allexport추가선택을 설정한다. 1과 같다. 모든 추가선택이 간략을 가지는것은 아니다(표 10-1에서 보여 준다.).
4. allexport추가선택을 해제한다. 2와 같다.

표 10-1. 추가선택

추가선택이름	지름	기능
allexport	-a	변수들이 자동적으로 넘겨 지도록 설정되게 한다.
bgnice		배경일감들이 보다 낮은 우선권으로 실행된다.
emacs		지령행편집을 위해 내부지령 emacs편집기를 사용한다.
errexit	-e	이것이 설정되면 지령이 0이 아닌 탈퇴상태(실패)를 되돌릴 때 ERR트랩프를 실행하고 탈퇴한다. 초기화파일은 읽을 때 설정되지 않는다.
gmacs		지령편집을 위해 내부지령 gmacs편집기를 사용한다.
markdirs		파일이름확장을 사용할 때 등록부이름뒤에 거꿀빗선(\ )을 붙인다.
monitor	-m	일감조종을 허락한다.
noclobber		방향바꾸기가 사용될 때 파일우에 겹쓰지 않게 한다.
noexec	-n	지령들을 읽지만 그것들을 실행시키지는 않는다. 스크립트문법을 검사하는데 사용한다. 대화형으로 실행될 때 사용되지 않는다.
noglob	-f	경로이름확장을 할수 없게 한다. 즉 통용기호를 해제한다.
nolog		리력파일에 함수정의를 기억시키지 않는다.
notify		배경일감이 끝날 때 사용자에게 통보한다(1988년이후 판본에서만).
nounset		설정되지 않은 변수를 확장할 때 오유를 현시한다.
privileged	f	설정되면 쉘은 .profile나 ENV파일을 읽지 않는다.
setuid		스크립트들과 함께 사용된다.
trackall		별명을 추적한다.
verbose	-v	오유수정을 위해 확장방식을 설정한다.
vi	-x	지령행편집을 위해 내부지령 vi편집기를 사용한다.
xtrace		오유수정을 위해 echo방식을 설정한다.

3. 재촉문

Korn쉘은 4개의 재촉문들을 제공한다. Korn쉘을 대화형으로 사용할 때 1차와 2차 재촉문들을 사용하는데 이 재촉문들은 변화시킬수 있다. 변수 PS1는 1차재촉문으로서 초기에 화페기호($)로 설정된다. 이것은 등록가입하여 지령입력을 대기할 때 나타난다. 변수 PS2는 2차재촉문으로서 초기에 >문자로 설정된다. 이것은 지령을 일부분을 입력하고 Enter를 눌렀을 때 나타난다. 1차재촉문과 2차재촉문을 변화시킬수 있다.

1차재촉문 $는 기정적인 1차재촉문이다. 이 재촉문을 변화시킬수 있다. 재촉문들은 보통 .profile에서 정의된다.

실례 10-4

1. $ **ps1="$(uname –n) ! $ "**
2. *jody 1141 $*

설명

1. 기정적인 1차재촉문은 $이다. PS1재촉문은 콤퓨터이름$(uname-n), 현재리력번호의 수자와 $로 재설정한다. 감탄부호는 현재리력번호이다(감탄부호를 인쇄하려면 PS1정의에서 2개의 감탄부호(!!)를 입력한다.).
2. 새로운 재촉문이 나타난다.

2차재촉문 PS2재촉문은 2차재촉문이다. 그 값은 표준오유(현시장치)에 현시된다. 이 재촉문은 지령을 완성하지 않고 행바꾸기건을 눌렀을 때 나타난다.

실례 10-5

1. $ print "Hello
2. > there"
3. *Hello*
   there
4. $
5. $ **ps2="--→ "**
6. $ **print "Hi**
7. **----→**
   ----→
   ----→ there"
   Hi
   there
   $

설명

1. 2중인용부호가 문자렬 "Hello 다음에 나타나야 한다.
2. 행바꾸기가 입력될 때 2차재촉문이 나타난다. 닫기2중인용부호가 입력될 때까지 2차재촉문이 현시된다.
3. print지령의 출력이 현시된다.
4. 1차재촉문이 현시된다.
5. 2차재촉문이 재설정된다.
6. 2중인용부호가 문자렬 "Hi 다음에 나타나야 한다.
7. 행바꾸기가 입력될 때 새로운 2차재촉문이 나타난다. 닫기2중인용부호가 입력될 때까지 2차재촉문이 현시된다.

4. 탐색경로

지령행에서 또는 쉘스크립트에서 입력된 지령을 실행하기 위해 Korn쉘은 PATH변수에 렬거된 등록부들을 탐색한다. PATH는 쉘이 왼쪽에서 오른쪽으로 탐색하는 옹근

두점으로 분리되는 등록부들의 목록이다. PATH에 있는 점은 현재작업등록부를 나타낸다. 지령을 경로에서 렬거된 임의의 등록부들에서 찾을수 없다면 쉘은 통보문 《ksh:filename: not found》를 표준오유로 보낸다. 경로를 .profile파일에 설정하는것이 좋다. 탐색처리를 빨리 진행하기 위해 Korn쉘은 추적별명들을 사용한다(420페지의 《추적별명》을 참고).

실례 10-6

```
$ echo $PATH
/home/gsa12/bin:/usr/ucb:/usr/bin:/usr/local/bin:.
```

설명

Korn쉘은 /home/gsa12/bin에서 기동하는 지령들을 탐색한다. 만일 지령이 거기에 없으면 /usr/ucb을 탐색하고 그다음에 /usr/bin, usr/local/bin 그리고 마지막에 점에 의해 나타내는 사용자의 홈등록부를 탐색한다.

dot지령 dot지령은 Korn쉘의 내부지령이다. 그것은인수로서 스크립트이름을 가진다. 스크립트는 현재쉘의 환경에서 실행된다. 자식프로쎄스는 기동되지 않는다. dot지령은 보통.profile파일이나 ENV파일중의 어느 하나가 변경되면 그것들을 재실행시키기 위해 사용한다. 실례로 등록가입한 다음에 어느 한 파일에서 한개 설정이 변화되였으면 점지령을 사용하여 탈퇴하고 다시 등록가입함이 없이 초기화파일을 재실행시킬수 있다.

실례 10-7

```
$ . .profile
$ . .kshrc
$ . $ENV
```

설명

자식프로쎄스는 보통 지령들이 실행될 때 기동된다. 점지령은 현재쉘에 있는 매 초기화파일.profile과 ENV파일(.kshrc)을 실행시킨다. 이 파일에 있는 국부변수와 전역변수는 이 쉘에서 정의된다. 그렇지 않으면 사용자는 이 파일들이 등록가입쉘에 대하여 실행되도록 하기 위해 탈퇴했다가 다시 등록가입해야 한다. 점지령은 이렇게 하지 않아도 되게 한다.

5. 지령행

등록가입한 다음에 Korn쉘은 기정으로 $기호를 1차재촉문으로 현시한다. 쉘은 지령해석기이다. 쉘은 대화형으로 실행될 때 말단으로부터 지령행들을 읽어 그것들을 단어들로 가른다. 한개 지령행은 공백(빈칸이나 타브)으로 분리되고 Enter건을 누를 때 발생하는 행바꾸기로 끝나는 한개이상의 단어들로 구성된다. 첫번째 단어는 지령이고 그다음 단어는 지령의 인수들이다. 지령은 ls나 pwd와 같은 UNIX의 실행프로그람, cd나 jobs와 같은 내부지령 또는 쉘스크립트일수도 있다. 지령에는 쉘이 지령행을 구문해석할 때 해석해야 하는 메타문자라고 하는 특수문자들이 있을수 있다. 지령행이 너무 길면 행바

꾸기문자앞에 거꿀빗선을 넣어 다음행에서 계속 입력할수 있다. 지령행이 끝날 때까지 2차재촉문이 나타난다.

지령처리순서 지령행에 있는 첫번째 단어는 실행되는 지령이다. 이 지령은 예약어, 특수내부지령, 편의프로그람, 함수, 스크립트 또는 실행프로그람이 될수 있다. 지령은 다음의 순서에 따라서 실행된다.[1]

1. 예약어(if, while, until과 같은)
2. 별명(typeset -f를 참고)
3. 내부지령
4. 함수
5. 스크립트와 실행프로그람

특수한 내부지령들과 함수들은 쉘에서 정의되므로 현재쉘로부터 실행될수 있다. 따라서 그 실행속도가 더 빨라 진다. ls와 pwd와 같은 스크립트와 실행프로그람들은 디스크에 기억되며 쉘은 PATH환경변수를 탐색하여 등록부계층에서 그것들을 찾아야 한다. 그다음 쉘은 스크립트를 실행하는 새로운 쉘을 파생시킨다. 현재 사용하는 지령의 형태를 알기 위하여 내부지령 whence -v 또는 그의 별명 type를 사용한다(실례 10-8에서 보여 준다.).

실례 10-8

$ **type print**
print is a shell builtin
$ **type test**
test is a shell builtin
$ **type ls**
ls is a tracked alias for /usr/bin/ls
$ **type type**
type is an exported alias for whence –v
$ **type bc**
bc is /usr/bin/bc
$ **type if**
if is a keyword

탈퇴상태 지령이나 프로그람은 끝나면 어미프로세스에 탈퇴상태를 되돌린다. 탈퇴상태는 0과 255사이의 값이다. 보통 프로그람이 탈퇴할 때 귀환된 상태가 령이면 그 지령은 실행에서 성공한것이다. 탈퇴상태가 령이 아닐 때 지령은 실패한것이다. Korn쉘의 상태변수 ?는 마지막으로 실행된 지령의 탈퇴상태값으로 설정된다. 프로그람의 성공 또

---

[1], 내부지령은 함수보다 먼저 실행된다. 따라서 이때 별명이 함수이름으로 정의되여야 한다(418페지의 《별명》을 참고). Korn쉘의 1994판본에서는 함수와 내부지령들의 처리순서가 반대로 되기때문에 이문제를 해결하였다.

는 실패는 그 프로그람을 작성한 프로그람작성자가 결정한다. 쉘스크립트에서 exit지령을 사용하여 탈퇴상태를 명백히 알수 있게 조종할수 있다.

실례 10-9

1. $ **grep “ellie” /etc/passwd**
   ellie:GgMyBsSJavd16s:9496:40:Ellie Quigley:/home/jody/ellie
2. $ **echo $?**
   0

3. $ **grep “nicky” /etc/passwd**
4. $ **echo $?**
   1
5. $ **grep “scott” /etc/passsswd**
   grep:/etc/passsswd: No such file or directory
6. $ **echo $?**
   2

설명

1. grep프로그람은 /etc/passwd파일에서 패턴ellie를 탐색하여 성공한다. /etc/passwd로부터 행이 현시된다.
2. ?변수는 grep지령의 탈퇴값으로 설정된다. 령은 성공을 나타낸다.
3. grep프로그람은 /etc/passwd파일에서 사용자 nicky를 찾을수 없다.
4. grep 프로그람은 패턴을 찾을수 없으면 탈퇴상태를 1로 되돌려 준다.
5. grep는 파일 /etc/passwd를 열수 없기때문에 실패한다.
6. grep는 파일을 찾을수 없으면 탈퇴상태 2를 되돌려 준다.

다중지령과 지령그룹화 지령행은 여러개의 지령들로 이루어 질수 있다. 매개 지령은 반두점에 의해 분리되고 지령행은 행바꾸기로 끝난다.

실례 10-10

$ **ls; pwd; date**

설명

행바꾸기가 나타날 때까지 지령들을 왼쪽에서부터 오른쪽으로 가면서 실행한다. 지령들을 묶어 모든 출력을 다른 지령에 파이프하거나 파일에로 방향바꾸기할수 있다.

실례 10-11

$ **(ls ; pwd; date) > outputfile**

설명

매 지령들의 출력을 outputfile이라고 하는 파일로 보낸다.

지령의 조건실행 조건실행할 때 2개의 지령문자렬들은 2개의 특수한 문자 &&와 ||에 의해 분리된다. 이 메타문자들의 오른쪽에 있는 지령은 왼쪽에 있는 지령의 탈퇴조건에 따라 실행되거나 실행되지 않는다.

실례 10-12

```
$ cc prgm1.c –o prgm1 && prgm1
```

설명

만일 첫번째 지령이 성공하면(탈퇴상태 0을 가지면) &&뒤에 있는 지령이 실행된다.

실례 10-13

```
$ccprog.c>&errllmail bob<err
```

설명

첫번째 지령이 실패하면(령이 아닌 탈퇴상태를 가지면) ||뒤에 있는 지령이 실행된다.

배경에서 지령 보통 지령을 실행할 때 그것은 전경에서 실행되고 재촉문은 지령실행이 끝날 때까지 나타나지 않는다. 지령이 끝날 때까지 기다리는것이 언제나 편리한것은 아니다. 지령행의 끝에 기호(&)를 붙이면 쉘은 쉘재촉문을 즉시에 복귀하고 그 지령을 배경에서 동시에 실행한다. 다른 지령을 시작하기 위해 기다리지 않아도 된다.

배경일감으로부터 출력은 그 일감이 처리될 때 현시장치에 보내진다. 따라서 배경에서 지령을 실행시키려면 그 지령의 출력을 파일에로 방향바꾸기하거나 인쇄기와 같은 다른 장치로 파이프하여 출력이 현재 하고 있는 동작을 방해하지 않게 하여야 한다.

실례 10-14

```
1. $ man ksh | lp&
2. [1] 1557
3. $
```

설명

1. Korn쉘에 대한 안내페지의 출력은 인쇄기로 파이프된다. 지령의 끝에 기호 &를 붙여 일감배경에 넣는다.
2. 2개의 수가 현시장치에 나타난다. 중괄호안에 있는 수는 이것이 배경에 넣어진 첫번째 일감라는것을 현시한다. 두번째 수는 이 일감의 프로쎄스식별번호인 PID이다.
3. Korn쉘재촉문이 즉시에 나타난다. 배경에서 프로그람이 실행되고 있는 동안 쉘은 전경에서 또 다른 지령을 대기한다.

6. 지령행리력

리력기구는 리력파일에서 지령행에 입력된 지령들을 번호를 붙여 기억한다. 리력파일로부터 지령을 다시 호출하여 그 지령을 다시 입력하지 않고도 재실행시킬수 있다. 내부지령 history는 리력목록을 현시한다. 리력파일에 대한 기정이름은 .sh_history이고 그것은 홈등록부에 놓인다.

ksh가 처음에 리력파일을 호출할 때 HISTSTZE변수는 몇개의 지령들이 리력파일로부터 호출될수 있는가를 나타낸다. 기정크기는 128이다. HISTFILE변수는 지령들이 기억되여 있는 지령리력파일(~/l.sh_history는 기정값이다.)의 이름을 지적한다. 리력파일은 등록가입대화조종(login session)에서 다음대화조종으로 넘어 간다. 이것을 지우기하지 않으면 매우 커진다. 리력지령은 fc -l지령에 미리 설정된 별명이다.

history지령/지령들을 재현시 내부지령 history지령은 이미전에 입력된 번호가 붙은 지령들을 현시한다. 그 지령은 현시를 조종하는 인수를 가질수 있다.

실례 10-15

```
1.  $ history              # Same as fc –1
     1 ls
     2 vi file1
     3 df
     4 ps -eaf
     5 history
     6 more /etc/passwd
     7 cd
     8 echo $USER
     9 set
     10 history
2.  $ history –n           # Print without line numbers
     ls
     vi file1
     df
     ps –eaf
     history
     more /etc/passwd
     cd
     echo $USER
     set
     history
     history –n

3.  $ history 8            # List from 8th command to present
      8    echo $USER
```

```
     9    set
    10    history
    11    history -n
    12    history 8

4.  $ history –3              # List this command and the 3 preceding it
    10   history
    11   history -n
    12   history 8
    13   history -3

5.  $ history –1 –5         # List last 5 commands, preceding this one
    13   history –3            # in reversed order.
    12   history 8
    11   history -n
    10   history
    9  set

6.  $ history                 # Print last 5 commands,preceding this one
   10     history             # in order.
   11     history -n
   12     history 8
   13     history -3
   14     history –1 –5


7. $   history             # (Different history list)
   78 date
   79 ls
   80 who
   81   echo hi
   82   history

8.  $ history ls echo     # Display from most recent ls command to
   79   ls                   # most recent echo command.
   80   who
        echo hi

9.  $ history –r ls echo           # -r reverses the list
81   echo hi
80   who
```

r지령으로 지령들을 재실행 r지령은 지령행에 마지막으로 입력된 지령을 재실행한다. r지령다음에 한개의 공백과 번호가 오면 그 번호에 있는 지령이 실행된다. 만일 r지령다

음에 한개 공백과 한개 문자가 오면 그 문자로 시작되는 제일 마지막지령이 실행된다. 그 어떤 인수도 없으면 r지령은 리력목록에 있는 제일 먼저 입력된 지령을 재실행한다.

실례 10-16

1. $ **r date**
   date
   Mon Feb 15 12:27:35 PST 2001

2. $ **r 3** *redo command number 3*
   df

Filesystem	*kbytes*	*used*	*avail*	*capacity*	*Mounted on*
/dev/sd0a	*7735*	*6282*	*680*	*90%*	*/*
/dev/sd0g	*144613*	*131183*	*0*	*101%*	*/usr*
/dev/sd2c	*388998*	*211395*	*138704*	*60%*	*/home.*

3. $ **r vi** *# Redo the last command that began with pattern vi.*

4. $ **r vi file1=file2** *# Redo last command that began with vi*
   # and substitute the first occurrence of
   # file1 with file2.

설명

1. 마지막지령 date가 재실행된다.
2. 리력파일에 있는 세번째 지령이 실행된다.
3. 문자렬 vi로 시작하는 제일 마지막지령이 실행된다.
4. 문자렬 file1이 문자렬 file2로 교체된다. 마지막지령 vifile1이 vifile2로 교체된다.

지령행편집 Korn쉘은 리력목록을 대화형으로 편집할수 있게 하는 2개의 내부편집기로서 emacs와 vi를 제공한다. vi편집기를 사용하려면 아래에 있는 set지령을 추가하여 이 행을 .profile파일에 넣는다. emacs편집기는 리력파일에 있는 행들을 한번에 한행씩 처리하고 내부지령 vi편집기는 한개이상의 행으로 이루어 진 지령들을 처리한다. vi를 설정하려면 다음과 같이 입력한다.

set -o vi

또는

VISUAL=vi

또는

EDITOR=/usr/bin/vi

만일 emacs를 사용하려면 다음과 같이 입력한다.

set -o emacs

또는

VISUAL=emacs

또는

EDITOR=/usr/bin/emacs

set -o vi는 VISUAL을 무시하고 VISUAL은 EDITOR를 무시한다는것을 명심하여야 한다.

vi내부편집기 리력목록을 편집하려면 Esc건을 누르고 본문을 우아래이동, 좌우이동, 지우기, 삽입, 변경을 진행하기 위해서 vi에서 사용하는 표준건들을 사용한다(표 10-2를 참고). 편집한 다음 Enter건을 누른다. 지령이 실행되여 리력목록의 아래부분에 추가된다. 리력파일에서 우로 올라 가려면 Esc건을 누른 다음 k건을 누른다.

표 10-2. vi지령

지 령	기 능
리력파일이동	
Esc k 또는 +	리력목록을 우로 이동시킨다.
Esc j 또는 -	리력목록을 아래로 이동시킨다.
G	리력파일의 첫번째 행으로 이동시킨다.
5G	문자렬에 대한 리력파일의 다섯번째 지령에로 이동한다.
/string	리력파일을 우로 탐색한다.
?	리력파일을 아래로 탐색한다.
한행우에서 이동	
H	한행우에서 왼쪽으로 이동한다.
L	한행우에서 오른쪽으로 이동한다.
B	한단어를 뒤로 이동시킨다.
E 또는 w	한단어를 앞으로 이동시킨다.
ˆ 또는 0	행에서 첫번째 문자가 시작되는 곳으로 이동한다.
$	행의 끝으로 이동한다.
vi로 편집	
aA	본문을 추가한다.
iI	본문을 삽입한다.
dd dw x	본문을 완충기(행, 단어 또는 문자)에 넣고 지운다.
cc C	본문을 변화시킨다.
U U	복귀한다.
yy Y	Yank(행을 완충기에 복사한다.)

(표계속)

지 령	기 능
P P	완충기에 복사되거나 지워 진 행을 그 행의 아래 또는 우에 놓는다.
R R	행우에서 본문의 한개 문자나 임의의 부분을 교체한다.

emacs편집기 리력파일을 뒤로 이동시키려면 ˆP를 누른다. 앞으로 이동시키려면 ˆN을 누른다. emacs편집지령들을 사용하여 본문을 변화시키거나 수정한 다음 Enter건을 누른다. 그러면 지령이 재실행된다(표 10-3).

표 10-3. emacs지령

지 령	기 능
리력파일이동	
Ctrl-P	리력파일을 우로 이동시킨다.
Ctrl-N	리력파일을 아래로 이동시킨다.
Ctrl-B	한개 문자를 뒤로 이동시킨다.
Ctrl-R	문자렬을 뒤방향으로 탐색한다.
Esc B	한개 단어를 뒤로 이동시킨다.
Ctrl-F	한개 문자를 앞으로 이동시킨다.
Esc F	한단어를 앞으로 입력시킨다.
Ctrl-A	행의 시작으로 이동시킨다.
Ctrl-E	행의 끝으로 이동시킨다.
Esc<	리력파일의 첫번째 행으로 이동시킨다.
Esc>	리력파일의 마지막행으로 이동시킨다.
Ctrl-U	행을 지운다.
Ctrl-Y	행을 되살린다.
Ctrl-K	유표로부터 끝행까지 지운다.
Ctrl-D	한개 문자를 지운다.
Esc D	앞방향의 한개 단어를 지운다.
Esc H	뒤방향의 한개 단어를 지운다.
Esc space	유표위치에 표식을 설정한다.
Ctrl-X Ctrl-X	표와 표식을 변화시킨다.
Ctrl-P Ctrl-Y	유표로부터 표식까지의 구역을 완충기에 넣고 (Ctrl-P) 기입한다(Ctrl-Y).

FCEDIT[2]와 편집지령 fc지령은 리력파일을 편집하기 위해서 사용자가 선택한 편집기를 호출하기 위한 FCEDIT변수(보통 .profile파일에서 설정된다.)와 함께 사용할수 있는 내부지령이다. 이것은 체계의 임의의 편집기로 될수 있다. FCEDIT변수는 자주 사용하는 편집기의 완전경로이름으로 설정된다. 만일 FCEDIT가 설정되지 않으면 fc지령을 입력할 때 기정편집기 /bin/ed가 호출된다.

FCEDIT변수는 선택된 편집기로 설정되여야 한다. 편집하려는 리력목록으로부터 많은 항목들을 지정할수 있다.

지령들을 편집한 다음 Korn쉘은 전체 파일을 실행시킨다.

#기호가 앞에 붙은 임의의 행들은 설명문으로 취급되여 실행되지 않는다. 설명문과 파일이름확장에 대하여 표 10-4에서 보여 준다.

실례 10-17

1. $ **FCEDIT=/usr/bin/vi**
2. $ **pwd**
3. $ **fc**

<Starts up the full-screen vi editor with the pwd command on line 1>

4. $ **history**
   1 date
   2 ls–1
   3 echo "hello"
   4 pwd

5. $ **fc -3 -1** *# Start vi, edit, write/quit, and execute*
   # last 3 commands.

설명

1. FCEDIT변수에 vi, emacs, textedit등과 같은 체계에 있는 임의의 UNIX 본문편집기에 대한 경로이름을 대입할수 있다. 만일 설정되지 않으면 ed편집기가 기정값으로 된다.
2. pwd지령이 지령행에 입력된다. 그것은 리력파일에 놓인다.
3. fc지령은 편집창문에서 제일 마지막으로 입력된 지령과 함께 편집기

[2] 1988년이후의 Korn쉘판본에서 FCEDIT변수를 HISTEDIT라고 하고 fc지령을 hist라고 한다.

(FCEDIT에서 설정된)가 호출되게 하였다. 만일 사용자가 편집하고 편집기를 해제하면 거기에 입력된 임의의 지령들이 실행된다.

4. history지령은 제일 마지막으로 입력된 지령들을 현시한다.
5. 편집기의 완충기에 있는 history파일의 마지막 3개의 지령들로 편집기를 기동하는데 fc지령이 사용된다.

7. 설명과 파일이름확장

파일이름확장은 사용자가 파일이름의 일부분을 입력하고 Esc건을 눌러 파일이름의 나머지부분을 완성할수 있게 하는 기구이다.

실례들에서 [Esc]는 Esc건을 표시한다.

표 10-4. Esc건과 파일이름확장

조합	결 과
command [Esc]#	#기호는 지령앞에 #기호를 붙인다. 즉 지령을 설명으로 된 리력목록에 놓는다. 지령이 실행되지 않는다.
command [Esc]_	밑선은 마지막지령의 마지막단어를 유표위치에 삽입한다.
command [Esc]2_	유표위치에 마지막지령의 두번째 단어를 삽입한다.
word [Esc]*	*는 현재 word를 일치되는 모든 파일로 교체한다.
word [Esc]\	\ 은 현재 word를 같은 문자들로 시작하는 첫번째 파일이름으로 교체한다. 즉 파일이름을 확장한다.
word [Esc]=	현재 word와 같은 문자로 시작되는 모든 파일이름들을 현시한다. 그것들은 번호가 붙은 목록으로 현시된다.

실례 10-18

(Press the Esc Key for [Esc].

1. $ **ls a[Esc]=**
   1) abc
   2) abc1
   3) abc122
   4) abc123
   5) abc2
2. $ **ls a[Esc]***
   ls abc abc1 abc122 abc123 abc2
   abc   abc1 abc122 abc123 abc2
3. $ **print apples pears peaches**
   apples pears peaches
4. $ **print [Esc]_**
   print peaches

peaches

5. $ **print apples pears peaches plums**
   apples pears peaches

6. $ **print [Esc]2_**
   print pears
   pears

설명

1. a, Esc건, 같기표식(=)을 차례로 입력하면 a로 시작되는 모든 파일들이 번호가 붙은 목록으로 현시된다(번호들은 실지로 그 어떤 특수한 의미도 가지지 않는다.).
2. a, Esc건, 별표(*)를 차례로 입력하면 a로 시작되는 파일이름들이 현시된다.
3. print지령은 인수 apples, pears, peaches들을 현시한다.
4. 밑선(_)앞에 놓여 있는 Esc건은 마지막인수로 교체된다.
5. print지령은 자기의 인수들인 apples, pears, peaches를 현시한다.
6. 수2와 밑선앞에 놓이는 Esc건은 두번째 인수로 교체된다. 지령(print)은 단어 0에서 시작하는 첫번째 인수이다.

8. 별명

별명은 지령에 대하여 Korn쉘이나 사용자가 정의한 략자이다. 별명은 자모, 수자, 문자들로 구성된다. 기정별명들은 쉘에 의해 제공되며 재정의되거나 해제될수 있다. C쉘의 별명들과 달리 Korn쉘은 인수넘기기를 지원하지 않는다(인수들을 사용할 필요가 있다면 451페지의 《함수》를 참고한다.). 별명들을 ENV파일에 기억시켜 자식쉘들에 넘길수 있다(ENV파일에 있는 지령들은 새로운 쉘이 파생될 때마다 실행된다.).

Korn쉘의 1988년판본에서 -x추가선택은 새로운 쉘이 ksh에 의해 개별적으로 호출되지 않는 한 별명들을 자식쉘들로 넘긴다. 쉘이 경로를 탐색하는 시간을 절약하기 위해 Korn쉘은 추적별명들을 사용한다. 별명들은 자기자체에 별명을 붙일수 있다. 즉 그것들은 재귀적이다.

별명목록제시 내부지령 alias는 설정된 모든 별명들의 목록을 현시한다.

실례 10-19

1. $ **alias**
2. autoload=typeset -fu
3. false=let 0
4. functions=typeset -f
5. hash=alias -t
6. history=fc -1
7. interger=typeset -i

8. r=fc -1
9. stop=kill -STOP
10. suspend=kill –STOP $$
11. true=:
12. type=whence –v

설명

1. alias지령은 인수가 없으면 모든 별명들을 표시한다. 이것은 이미 설정된 별명들의 목록이다.
2. autoload별명은 함수들을 동적으로 호출하는데 사용된다.
3. false별명은 거짓조건을 검사하는 식에서 사용된다.
4. functions별명은 모든 함수들과 정의들을 표시한다.
5. hash별명은 모든 추적별명들을 표시한다.
6. history별명은 번호가 앞에 붙은 리력파일 .sh_history에 있는 모든 지령들을 표시한다.
7. integer별명은 옹근수형변수들을 작성하게 한다.
8. r별명은 리력목록으로부터 이전의 지령을 재실행하게 한다.
9. stop별명은 일감번호나 PID가 kill지령에 제공되면 프로쎄스가 중지되게 한다. 일감은 fg를 입력하여야 전경에서 다시 실행될수 있다.
10. suspend별명은 STOP신호와 프로쎄스 PID($$)를 kill지령에 보내여 현재 일감을 중단시킨다.
11. true별명은 아무것도 하지 않는 지령에 설정되는데 보통 무한순환을 시작하는데 사용된다.
12. type별명은 별명, 2진실행형 등과 같은 현재 실행하고 있는 지령의 형태를 가리킨다.

별명만들기 사용자는 Korn쉘에서 별명을 작성할수 있다. 별명은 존재하는 지령이나 지령들에 대한 가명이다. 실지 지령들은 쉘이 지령행을 해석할 때 별명대신 교체된다.

실례 10-20

1. $ **alias cl='clear'**
2. $ **alias l='ls -laF'**
3. $ **alias ls='ls -aF'**
4. $ \ **ls ..**

설명

1. 별명 cl은 Clear에 대한 별명이다.
2. 별명은 l이다. 문자l은 지령 ls –laF에 대한 가명이다.
3. 별명 ls에 지령 ls –aF가 대입된다.
4. 거꿀빗선은 행을 실행하기 위해 별명의 의미를 해제한다. 별명이 아닌 실지 ls지령이 실행된다.

별명지우기 unalias지령은 별명을 지우기한다.

실례 10-21

```
unalias cl
```

설명

별명 cl이 설정된 별명들의 목록에서 지워 진다.

추적별명 경로를 탐색하는데 필요한 시간을 줄이기 위해 Korn쉘은 어떤 지령이 처음으로 나타날 때 별명을 작성하고 그것을 그 지령의 완전경로이름으로 설정한다.

이것을 추적별명이라고 한다.[3] Korn쉘은 설치될 때 미리 정의된 몇개의 추적별명들을 가지고 있다. 추적별명들을 사용하기 위해 set -o trackall지령이 제공된다. 그것은 보통 ENV파일에서 설정된다. 모든 추적별명들을 보려면 alias -t를 입력한다.

실례 10-22

```
$ alias –t
chmod=/bin/chmod
ls=bin/ls
vi=/usr/ucb/vi
who=/bin/who
```

설명

내부지령 alias에 대한 -t추가선택은추적기구를 통해 별명이 붙여 진 지령들을 현시한다. 사용자가 이 지령들을 입력할 때 쉘은 그 경로를 탐색하지 않고 별명정의를 사용하여 그 지령을 호출한다.

9. 일감조종

일감조종은 배경과 전경일감들의 실행조종에 사용된다. Korn쉘의 일감조종을 사용하려면 일감조종을 지원하지 않는 체계에서 monitor추가선택(set-o monitor)을 설정해야 한다. 표 10-5에서 일감조종지령들에서 보여 준다.

실례 10-23

```
1.  $ vi
    [1]   + Stopped              #vi

2.  $ sleep 25&
    [2] 4538

3.  $ jobs
    [2] + Running              # sleep 25&
    [1] – Stopped              #vi
```

3. 추적별명들은 PATH변수가 재설정되면 정의가 해제된다.

4. $ jobs -1
   [2] + 4538   Running   # sleep 25&
   [1] – 4537   Stopped   # vi
5. $ fg %1

설명

1. vi편집기가 호출된후 ^Z(Control-Z)를 눌러 vi대화조종을 중단시킬수 있다. 편집기는 배경에서 중단되고 통보 Stopped가 나타난 다음 쉘재촉문이 즉시에 나타난다.
2. 지령끝에 있는 &는 인수25를 가진 sleep지령이 배경에서 실행되게 한다. 표기 [2]는 이것이 배경에서 실행되는 두번째 일감이라는것을 표시하며 4538은 이 일감의 PID이다.
3. jobs지령은 현재배경이 있는 일감들을 현시한다.
4. -l추가선택을 가진 jobs지령은 배경에서 실행되는 프로쎄스들(일감들)과 일감들의 PID번호를 현시한다.
5. 기호 %와 일감번호가 뒤에 붙은 fg지령은 그 번호의 일감을 전경으로 넘긴다. 번호가 없으면 fg는 가장 최근의 배경일감을 전경으로 넘긴다.

표 10-5. 일감조종지령

지 령	의 미
Jobs	끝나지 않은 모든 프로쎄스들을 일감의 번호가 중괄호 안에 순서대로 배렬된 목록으로 현시한다.
jobs-l	jobs와 같지만 일감의 PID번호를 포함한다.
^Z	현재일감을 정지시킨다.
fg%n	전경에서 배경일감을 실행시킨다.
bg%n	배경에서 일감을 실행시킨다.
wait%n	번호가 n인 일감이 끝나기를 기다린다.
kill%n	번호가 n인 일감을 중지시킨다.

10. 메타문자

메타문자들은 자기자체가 아닌 다른것을 표시하는데 사용되는 특수문자들이다. 표 10-6에서는 일반적인 몇개의 Korn쉘메타문자들과 기능을 보여 준다.

표 10-6. 쉘메타문자

지 령	기 능
\	다음문자의 문자(literal)해석
&	배경처리

(표계속)

지 령	기 능
;	지령분리기
$	변수바꿔넣기
?	한 문자 대조
[abc]	문자모임에 있는 한개 문자를 대조한다.
[!abc]	문자모임에 있지 않는 한개 문자를 대조한다.
*	임의의 수의 모든 문자들을 대조한다.
(cmds)	자식쉘에서 지령들을 실행한다.
{cmds}	현재쉘에서 지령들을 실행한다.

실례 10-24

```
1.  $ ls –d *     all files are displayed
    abc   abc122   abc2   file1.bak file2.bak nonsense   nothing one
    abc1 abc123    file1    file2     none      noone

2.  $ print hello \          # The carrage return is escaped
    > there
    hello there

3.  $ rusers&                 # Process the rusers command in the background
    [1]     4334

    $
4.  $ who; date; uptime       # Commands are executed one at a time
    ellie   console Feb 10 10:46
    ellie   ttyp0     Feb 15 12:41
    ellie   ttyp1     Feb 10 10:47
    ellie   ttyp2     Feb 5    10:53
    Mon Feb 15 17:16:43 PST 2001
    5:16pm up 5 days, 6:32,1 uaer, load average: 0.28, 0.23, 0.01

5.  $ print $HOME           # The value of the HOME variable is printed
   /home/jody/ellie

6.  $ print $LOGNAME      # The value of the LOGNAME variable is printed
    ellie

7.  $ ( pwd; cd / ; pwd )
    /home/jody/ellie
    /
    $ pwd
```

```
   /home/jody/ellie
8. $ ( pwd; cd / ; pwd )
   /home/jody/ellie
   /
   $ pwd
   /
9. $ ( date; pwd; ls ) > outfile
   $ cat outfile
   Mon Feb 15 15:56:34 PDT 2001
   /home/jody/ellie
   foo1
   foo2
   foo3
```

설명

1. 별표는 현재등록부에 있는 모든 파일들을 대조한다(ls지령에 대한 -d추가선택은 부분등록부의 내용이 현시되지 않게 한다.).
2. 거꿀빗선은 지령행이 다음행에서 계속되도록 한다.
3. 기호 &는 rusers프로그람이 배경에서 실행되도록 한다. 즉 쉘재촉문이 즉시에 복귀된다. 두 프로쎄스들이 동시에 실행된다.
4. 매 지령은 반두점에 의해 분리된다. 매 지령은 한번에 하나씩 실행된다.
5. 표식 $가 변수앞에 붙으면 쉘은 변수바꿔넣기를 진행한다. env변수의 값 HOME이 현시된다. HOME변수에 사용자홈등록부의 완전경로이름이 있다.
6. 표식 $가 변수이름앞에 다시 붙는다. LOGNAME변수의 값은 사용자등록가입이름이다.
7. 괄호는 괄호안에 있는 지령들이 자식쉘에서 실행된다는것을 표시한다. cd지령이 쉘에 내장되여 있으므로 호출되는 매 쉘은 자기의 cd사본을 가지고 있다. pwd지령은 현재작업등록부 /home/jody/ellie를 인쇄한다. 뿌리등록부를 변경시킨 다음 pwd지령은 새로운 등록부가 root라는것을 현시한다. 자식쉘이 탈퇴한후 어미쉘에 있는 pwd지령의 출력은 등록부가 여전히 자식쉘에 들어 가기전과 같이 /home/jody/ellie로 설정되여 있다는것을 나타낸다.
8. 대괄호는 그 안에 있는 지령들이 현재쉘에서 실행된다는것을 현시한다. pwd지령은 현재작업등록부 /home/jody/ellie를 인쇄한다. 뿌리등록부로 등록부를 변경시킨후 pwd지령은 새로운 등록부가 root라는것을 현시한다. 대괄호안에 있는 지령들이 끝난후 pwd지령의 출력은 등록부가 여전히 홈등록부로 설정되여 있다는것을 표시한다.
9. 괄호는 3개의 지령들의 출력을 outfile에 보내기 위해 지령들을 묶는데 사용한다.

11. 파일이름바꿔넣기(통용기호)

지령행을 해석할 때 쉘은 메타문자들을 사용하여 흔히 통용기호라고 하는 문자들의 어떤 모임과 대응되는 파일이름이나 경로이름들을 간략화한다.

표 10-7에서 보여 준 파일이름바꿔넣기메타문자들은 파일이름의 자모문자모임으로 확장된다.

문자를 파일이름으로 확장하는 과정을 파일이름바꿔넣기라고 한다. 만일 메타문자가 사용되여 그것과 대응되는 파일이름이 없다면 Korn쉘은 메타문자를 문자그대로 취급한다.

표 10-7. 쉘메타문자들과 파일이름바꿔넣기

메타문자	의 미
*	빈값 혹은 모든 문자들을 대신한다.
?	한개 문자만을 대신한다.
[abc]	문자모임에 있는 한개 문자 a, b, c를 대신한다.
[a-z]	이 범위에 있는 한개 문자를 대신한다. 즉 a와 z사이의 모임에 있는 임의의 물음표를 대신한다.
~	사용자의 홈등록부로 ~를 치환한다.
\	메타문자로 사용되지 않게 한다.

별표 별표는 파일이름에 있는 빈값 혹은 모든 문자들을 대신하는 통용기호이다.

실례 10-25

1. $ **ls ***
   abc abc1 abc122 abc123 abc2 file1 file1.bak file2 file2.bak none nonsense noone nothing nowhere one

2. $ **ls *.bak**
   file1.bak file2.bak

3. $ **print a*c**
   abc

설명

1. 별표는 현재작업등록부에 있는 모든 파일들을 의미한다. 모든 파일들은 ls에 넘겨 져 현시된다.
2. 빈값 혹은 여러개의 문자들로 시작되고 .bak로 끝나는 모든 파일들이 대응되여 현시된다.
3. a로 시작되고 그다음에 여러개의 물음표들이 오고 마지막에 c로 끝나는 모든 파일들이 대응되며 인수로서 print지령에 넘겨 진다.

물음표 물음표는 파일이름에서 한개 문자를 나타낸다. 파일이름에 한개이상의 물음표가 있으면 쉘은 그 물음표를 파일이름에서 대응되는 문자로 치환하여 파일이름바꿔넣기를 진행한다.

실례 10-26

1. $ ls
   abc abc1 abc122 abc123 abc2 file1 file1.bak file2 file2.bak
   none nonsense noone nothing nowhere one
2. $ ls a?c?
   abc1 abc2
3. $ ls ??
   ?? not found
4. $ print abc???
   abc122 abc123
5. $ print ??

설명

1. 현재등록부에 있는 모든 파일들을 표시한다.
2. 파일이름이 a로 시작되고 그다음에 한개 문자가 놓이며 그리고 c 그다음에 한개 문자가 더 놓이는 4개의 문자가 포함된 파일이름을 대조하여 표시한다.
3. 2개의 문자만을 포함하는 파일이름들이 표시된다. 그런것이 하나도 없으므로 2개의 물음표는 문자그대로 취급된다.등록부에 ??라는 파일이 없으므로 쉘은 통보문《?? not found》를 내보낸다.
4. 6개의 문자를 포함하는 abc로 시작되고 그다음에 임의의 3개문자가 놓이는 파일이름들을 대조하여 인쇄한다.
5. ksh print함수는 2개의 물음표를 인수로 받는다. 쉘은 2개 문자만으로 된 파일이름들을 찾는다. 등록부에 2개 문자만을 포함하고 있는 파일은 없다. 일치되는 파일이름이 없으면 쉘은 물음표를 문자그대로 취급한다. 2개의 물음표는 인수로서 print지령에 넘겨 진다.

중괄호 중괄호는 문자들의 모임이나 범위에서 한개 문자만을 포함하는 파일이름들을 대응시키는데 쓰인다.

실례 10-27

1. $ ls
   abc abc1 abc122 abc123 abc2 file1 file1.bak file2 file2.bak
   none nonsense noone nothing nowhere one
2. $ ls abc[123]
   abc1 abc2
3. $ ls abc[1-3]
   abc1 abc2
4. $ ls [a-z][a-z][a-z]
   abc one
5. $ ls [!f-z]???
   abc1 abc2

6. $ ls abc12[2-3]
   abc122 abc123

설명

1. 현재작업등록부에 있는 모든 파일들이 표시된다.
2. abc로 시작되고 그다음에 1, 2, 3중에서 어느 하나가 오는 4개 문자로 된 모든 파일이름들이 대조되여 현시된다. 괄호안에 있는 모임에서 한개 문자만이 파일이름과 대조된다.
3. abc로 시작되고 그다음에 1부터 3사이의 한개의 수가 오는 4개 문자로 된 모든 파일이름들을 대조하여 표시한다.
4. 3개 소문자의 자모문자가 있는 3개문자로 된 모든 파일이름이 대조되여 표시된다.
5. 첫번째 문자가 f와 z사이에 있는 문자가 아니고 그다음에 3개의 임의문자(???)가 놓이는 네문자로 된 모든 파일들이 표시된다.
6. 파일이름이 abc12로 시작되고 그다음에 2 또는 3이 놓이는 파일이 표시된다.

메타문자탈출 메타문자를 문자그대로 사용하자면 거꿀빗선을 사용하여 메타문자가 해석되지 않게 한다.

실례 10-28

1. $ **ls**
   abc file1 youx
2. $ **print How are you?**
   How are youx
3. $ **print How are you\ ?**
   How are you?
4. $ **print When does this line**
   > ever end\ ?
   When does this line ever end?

설명

1. 현재작업등록부에 있는 파일들이 표시된다. 파일 youx를 주목해야 한다.
2. 쉘은 물음표에서 파일이름확장을 한다. you로 시작되고 그다음에 한개문자만이 놓이는 현재등록부에 있는 임의의 파일이 대조되여 문자렬에 대입된다. 파일이름 youx가 문자렬에 대입되여 문자렬이 How are youx로 된다(이것은 바라지 않던 결과이다.).
3. 물음표앞에 거꿀빗선을 붙이면 물음표가 메타문자로부터 벗어나 쉘이 그것을 통용기호로 해석하지 않는다.
4. 행바꾸기앞에 거꿀빗선을 붙여 그것이 메타문자로부터 벗어 나게 한다. 2차재촉문은 문자렬이 행바꾸기문자로 끝날 때까지 현시된다. 물음표(?)가 메타문자에서 벗어나 파일이름확장에 사용되지 않는다.

물결표(~)와 련결부호(~~)확장 물결표는 경로이름확장을 위해 Korn쉘에 (C쉘로부터) 도입되였다. 물결표는 단독으로 쓰일 때 사용자홈등록부의 완전경로이름과 같다.

물결표가 사용자이름에 추가되면 그것은 그 사용자의 완전경로이름으로 확장된다.

련결부호문자는 이전의 작업등록부를 표시한다. OLDPWD도 이전의 작업등록부를 표시한다.

실례 10-29

1. $ **echo ~**
   /home/jody/ellie
2. $ **echo ~joe**
   /home/joe
3. $ **echo ~+**
   /home/jody/ellie/perl
4. $ **echo ~-**
   /home/jody/ellie/prac
5. $ **echo $OLDPWD**
   /home/jody/ellie/prac
6. $ **cd -**
   /home/jody/ellie/prac

설명

1. 물결표는 사용자홈등록부의 완전경로이름과 같다.
2. 사용자이름앞에 물결표가 붙으면 그것은 joe의 홈등록부의 완전경로이름과 같다.
3. ~+표기는 작업등록부의 완전경로이름과 같다.
4. ~—는 이전 작업등록부와 같다.
5. OLDPWD변수는 이전의 작업등록부를 포함한다.
6. 련결부호는 이전의 작업등록부를 표시한다. 즉 이전의 작업등록부로 등록부를 변경시켜 그 등록부를 현시한다.

새로운 ksh메타문자 새로운 Korn쉘메타문자들은 egrep와 awk의 정규표현메타문자들과 류사하게 파일이름확장에 사용된다. 괄호안에 있는 문자앞에 붙은 메타문자들은 패턴대조를 조종한다. 표 10-8에서 보여 준다.

실례 10-30

1. $ **ls**
   abc abc1 abc122 abc123 abc2 file1 file1.bak file2 file2.bak none nonsense noone nothing nowhere one
2. $ **ls abc?(1|2)**
   abc abc1 abc2
3. $ **ls abc*([1-5])**

abc   abc1   abc122   abc123 abc2

4. **$ ls abc+([1-5])**
   abc1   abc122 abc123 abc2

5. **$ ls no@(thing|ne)**
   none   nothing

6. **$ ls no!(one|nsense)**
   none   nothing nowhere

설명

1. 현재작업등록부에 있는 모든 파일들이 표시된다.
2. abc로 시작되고 그다음에 령문자가 오거나 괄호안에 있는 패턴중에서 어느 하나가 오는 파일이름들을 대조한다. abc, abc1, 또는 abc2와 대조한다.
3. abc로 시작되고 1과 5사이에 있는 임의의 수자들이 오는 파일이름들을 대조한다. abc, abc1, abc122, abc123, abc2와 대조한다.
4. abc로 시작되고 그다음에 1과 5사이에 있는 한개이상의 수자들이 오는 파일이름들을 대조한다. abc1, abc122, abc123, abc2를 대조한다.
5. no로 시작되고 그다음에 thing이나 ne가 오는 파일이름들과 대조한다. nothing이나 none을 대조한다.
6. no로 시작되고 그다음에 one 또는 nsense를 제외한 임의의것이 오는 파일이름들과 대조한다. none, nothing 그리고 nowhere를 대조한다.

표 10-8. 정규식통용기호

정규식	의 미
abc?(2|9)	?는 괄호안에 있는 임의의 패턴중에서 어느 하나를 대조하거나 아무것도 대조하지 않는다. 수직선은 or조건을 표시한다(실례 2 또는 9). abc21, abc91 또는 aabc1을 대조한다.
abc*([0-9])	*는 괄호안에 있는 임의의 패턴중에서 임의의 수의 요소들을 대조한다. 뒤에 임의의 수의 수자들이 오는 abc를 대조한다. 실례 abc, abc1234, abc3, abc2 등등
a bc+([0-9])	+는 괄호안에 있는 임의의 패턴중에서 한개이상의 인수들을 대조한다. 그뒤에 한개이상의 수자들이 오는 abc를 대조한다. 실례 abc3, abc123 등등
no@(one|ne)	@는 괄호안에 있는 임의의 패턴중에서 한개 인수만을 대조한다. noone나 none를 대조한다.
no!(thing|where)	!는 괄호안에 있는 임의의 패턴과 대조되는것을 제외한 모든 문자렬을 대조한다. no, nobody 또는 noone을 대조하지만 nothing이나 nowhere는 대조되지 않는다.

noglob변수 noglob변수가 설정되면 파일이름바꿔넣기를 하지 못한다. 즉 모든 메타문자들이 자기자체들을 현시하고 통용기호로 사용되지 않는다. 이것은 grep, sed 또는 awk와 같은 프로그람에서 메타문자들을 포함하고 있는 패턴들을 탐색할 때 사용할수

있다. noglob가 설정되지 않을 때 모든 메타문자들이 해석되지 않게 하려면 거꿀빗선으로 탈퇴시켜야 한다.

실례 10-31

1. % **set –o noglob** *# or set –f*
2. % **print * ?? [] ~ $LOGNAME**
   ** ?? [] /home/jody/ellie ellie*
3. % **set +o noglob** *# or set +f*

설명

1. noglob변수가 설정된다. 이것은 파일이름확장을 위해 통용기호가 자기의 특수한 의미로 쓰이지 않게 한다. -f추가선택을 사용하여 지령을 설정하면 같은 결과를 얻을수 있다.
2. 파일이름확장문자들이 해석되지 않고 그자체가 현시된다. 물결표와 표식 $가 확장된다는것을 주목해야 한다.
3. noglob추가선택이 재설정된다. 파일이름메타문자들이 확장된다.

12. 변수

국부변수 국부변수들은 그것이 작성된 쉘에게만 알려 지는 주어 진 값들이다. 변수이름들은 자모문자나 밑선문자로 시작되여야 한다. 나머지문자들은 자모문자나 0부터 9까지의 아라비아수자 또는 밑선문자가 될수 있다. 변수이름의 끝에는 임의의 다른 문자들이 올수 있다.

국부변수의 설정과 참조 값을 변수에 대입할 때 같기부호량옆에 공백이 없어도 된다. 변수를 빈값으로 설정하려면 같기부호다음에 아무것도 없으면 된다. 한개이상의 단어가 변수에 대입되면 그것을 인용부호안에 넣어 공백을 보호해야 한다. 그렇지 않으면 쉘은 오유통보를 인쇄하고 그 변수는 정의되지 않는다. 표식 $가 변수이름앞에 붙으면 그 변수에 대입된 값을 참조할수 있다. 만일 다른 문자들이 변수이름에 붙으면 대괄호를 사용하여 변수이름들을 다른 문자들로부터 분리한다.

실례 10-32

1. $ **state=Cal**
   $ echo $state
   Cal
2. $ **name="Peter Piper"**
   $ echo $name
   Peter Piper
3. $ **x=**
   $ echo $x
   # Blank line appears when a variable is either unset or set to null
   $

4. $ **state=Cal**
   $ **print ${state}ifornia**
   California

설명

1. 변수 state에 값 Cal이 대입된다. 쉘은 표식 $가 앞에 붙은 변수이름을 발견하면 변수바꿔넣기를 진행한다. 변수의 값이 현시된다.
2. 변수 name에 값 《Peter Piper》가 대입된다. 쉘이 지령행을 해석할 때 문자렬을 개개의 단어들로 나누지 않도록 공백을 숨기자면 문자렬을 인용부호안에 넣어야 한다. 변수의 값이 현시된다.
3. 변수 x에 값이 아니라 빈문자렬이 대입된다. 빈값 즉 빈문자렬이 현시된다. 변수가 전혀 설정되지 않았다면 같은 결과가 현시된다.
4. 변수 state에 값 Cal이 대입된다. 변수를 대괄호안에 넣어 거기에 붙은 문자들로부터 보호한다. 변수 Cal의 값에 ifornia가 추가되여 현시된다.

국부변수들의 유효범위 국부변수들은 그것이 작성된 쉘에서만 유효하며 그것들은 자식쉘들로 넘겨 질수 없다. $$변수는 현재쉘의 PID(프로쎄스식별번호)를 포함하는 특수변수이다.

실례 10-33

1. $ **echo $$**
   1313
2. $ **round=world**
   $ **echo $round**
   world
3. $ **ksh**   *# Start a subshell*
4. $ **echo $$**
   1326
5. $ **echo $round**
6. $ **exit**   *# Exits this shell, returns to parent shell*
7. $ **echo $$**
   1313
8. $ **echo $round**
   world

설명

1. $$변수의 값은 현재쉘의 PID와 같다. 이 쉘의 PID는 1313이다.
2. 국부변수 round에 문자렬값 world가 대입되고 그 변수의 값이 현시된다.
3. 새로운 Korn쉘이 호출된다. 이것을 보조쉘 또는 자식쉘이라고 한다.
4. 이 쉘의 PID는 1326이다. 어미쉘의 PID는 1313이다.
5. 변수 round는 이 쉘에서 정의되지 않는다.
6. exit지령은 이 쉘을 끝내고 어미쉘로 되돌아 간다. 만일 ignoreeof추가선택이 설정되지 않으면 Control-D로서 이 쉘을 끝낸다.

7. 어미쉘이 복귀되고 그의 PID가 현시된다.
8. 변수의 값이 현시된다.

읽기전용변수설정 읽기전용변수는 재정의하거나 해제할수 없다. 이것은 내부지령들인 readonly나 typeset-r로 설정할수 있다. 우선권방식에서 실행할 때 안전성을 목적으로 하여 변수들을 읽기전용변수로 설정하려 할수 있다.

실례 10-34

1. $ **readonly name=Tom**
   $ **print $name**
   Tom
2. $ **unset name**
   ksh: name: is read only
3. $ **name=Joe**

   ksh name: is read only
4. $ **typeset –r PATH**
   $ **PATH=${PATH}:/usr/local/bin**
   ksh: PATH: is read only

설명

1. 국부변수 name에 값 Tom이 대입된다.
2. 읽기전용변수는 해제할수 없다.
3. 읽기전용변수는 재정의할수 없다.
4. PATH변수는 읽기전용으로 설정된다. 이 변수를 해제하거나 변화시키려고 하면 오유통보가 나타난다.

환경변수 환경변수들은 그것들을 작성한 쉘과 그 쉘로부터 파생된 임의의 자식쉘이나 프로쎄스에서 다 사용할수 있다. 보통 환경변수를 대문자로 쓴다. 변수가 작성된 쉘은 어미쉘이라고 한다. 새로운 쉘이 어미쉘로부터 파생된다면 그것을 자식쉘이라고 한다. HOME, LOGNAME, PATH, SHELL과 같은 일부환경변수들은 /bin/login프로그람에 의해 등록가입하기전에 설정된다. 보통 환경변수들은 사용자의 홈등록부에 있는 .profile 파일에서 설정된다.

환경변수설정 환경변수를 설정하려면 값을 대입한 다음에 변수가 설정될 때 export지령을 사용한다. 스크립트에 있는 모든 변수들은 set지령에 allexport추가선택을 설정하여 반출할수 있다.

실례 10-35

1. $ **TERM=wyse ; export TERM**
2. $ **export NAME = "John Smith"**
   $ **print $NAME**
   John Smith

3. $ **print $$**
   319
4. $ **ksh**
5. $ **print $$**
   340
6. $ **print $NAME**
   John Smith
7. $ **NAME=April Jenner”**
   $ **print $NAME**
   April Jenner
8. $ **exit**
9. $ **print $$**
   319
10. $ **print $NAME**
    John Smith

설명

1. TERM변수에 wyse가 대입되여 반출된다. 따라서 이 쉘로부터 파생된 프로세스들은 변수를 계승한다.
2. 변수가 반출되고 같은 단계에서 정의된다(Korn쉘과 함께 새로 시작되는 변수이다.).
3. 쉘의 PID값이 인쇄된다.
4. 새로운 Korn쉘이 시작된다. 새로운 쉘을 자식이라고 한다. 원래쉘은 이 쉘의 어미이다.
5. $$(340)변수에 기억된 새로운 Korn쉘의 PID가 인쇄된다.
6. 변수가 새로운 쉘에 넘겨 지고 현시된다.
7. 변수가 April Jenner로 재설정되고 현시된다.
8. Korn자식쉘이 탈퇴하여 어미쉘에 복귀한다.
9. 어미의 PID인 319가 다시 현시된다.
10. 변수 NAME은 자기의 초기값을 가지고 있다. 변수들은 어미에서 자식쉘로 넘겨 질 때 자기의 값을 그대로 유지한다. 자식은 그의 어미에서 설정된 변수의 값을 변화시킬수 없다

특수환경변수 korn쉘은 기정값을 환경변수들인 PATH, PS1, PS2, PS3, PS4, MAIL CHECK, FCEDIT, TMOUT, IFS에 대입한다. SHELL, LOGNAME, USER, HOME은 /bin/login프로그람에 의해 설정된다. 기정값들을 변화시킬수 있으며 표 10-9에 있는 다른값들은 설정할수 있다.

표 10-9. **Korn쉘환경변수**

변수이름	의 미
_(밑선)	이전 지령의 마지막인수이다.
CDPATH	지령에 대한 탐색경로이다. 만일 경로이름이 /, ./ 또는 ../로 시작되지 않을 때 등록부를 찾는데 사용되는 등록부들의 웅근두점으로 분리되는 목록

(표계속)

변수이름	의 미
COLUMNS	설정되면 쉘편집방식과 추가선택지령을 위한 편집창문의 너비를 정의한다.
EDITOR	내부 편집기 emacs, gmacs 또는 vi에 대한 경로이름
ENV	ksh프로그람이 호출될 때 Korn쉘이 호출하는 파일의 이름으로 설정된 변수이다. 이 파일에는 함수들과 별명들이 있다. 1988년이후 판본에서 이 파일은 ksh가 대화형으로 호출될 때에만 실행되고 비대화형쉘에 대해서는 실행되지 않는다. 만일 우선권추가선택이 설정되면 변수가 확장되지 않는다.
ERRNO	체계오유번호이다. 그 값은 가장 최근에 실패한 체계호출의 오유번호이다.
FCEDIT	fc지령에 대한 기정편집기이름. 1988년이후 판본에서 이 변수는 HISTEDIT라고 하고 fc지령은 hist라고 한다.
FPATH	자동넣기되는 함수들을 가지는 등록부들에 대한 탐색경로를 정의하는 등록부들의 옹근두점으로 분리된 목록
HISTEDIT	1988년이후 Korn쉘판본에서 FCEDIT에 대한 새로운 이름이다.
HISTFILE	지령 history를 기억시킬 파일을 지정한다.
HISTSIZE	리력으로부터 호출될수 있는 지령들의 최대수, 기정값은 128이다.
HOME	홈등록부, 등록부가 지정되지 않을 때 cd에 의해 사용된다.
IFS	지령바꿔넣기, 순환명령구조에서 목록들 그리고 입력을 읽을 때 단어들을 분리시키기 위해 사용되는 SPACE, TAB, NEWLINE과 같은 내부마당분리기이다.
LINENO	스크립트에 있는 현재행번호이다.
LINES	차림표항목들을 수직으로 현시하기 위한 추가선택순환에 사용된다. 기정값은 24이다.
MAIL	이 파라메터가 우편파일이름으로 설정되고 MAILPATA파라메터가 설정되지 않으면 쉘은 사용자에게 지정된 파일에 우편이 도착했다는것을 알려 준다.
MAILCHECK	이 파라메터는 쉘이 MAILPATH나 MAIL파라메터들에 의해 지정된 파일들에 우편이 도착했는가를 몇번(초로) 검사하는가를 지정한다. 기정값은 600초(10분)이다. 만일 0으로 설정되면 쉘은 1차재촉문을 현시하기전에 검사한다.
MAILPATH	옹근두점으로 분리된 파라메터들의 목록이다. 만일 파라메터가 설정되면 쉘은 사용자에게 지정된 파일에 우편이 도착했다는것을 알려 준다. 매 파일이름뒤에는 %와 수정시간이 변할 때 인쇄되는 통보가 올수 있다. 기정통보는 You have mail이다.
OLDPWD	마지막작업등록부이다.
PATH	지령들에 대한 탐색경로이다. 실행하려고 하는 지령을 탐색하기 위해 쉘이 사용하는 옹근두점으로 분리된 등록부들의 목록이다.

(표계속)

변수이름	의 미
PWD	현재작업등록부이다. cd에 의해 설정된다.
PPID	어미프로쎄스의 프로쎄스ID이다.
PS1	1차재촉문문자렬, 기정문자로 $이다.
PS2	2차재촉문문자렬, 기정문자로 >이다.
PS3	추가선택지령과 함께 사용되는 추가선택재촉문문자렬, 기정으로 #?이다.
PS4	추적이 설정될 때 사용되는 오유수정재촉문문자렬, 기정으로 +이다.
RANDOM	변수가 참조될 때마다 생기는 우연수이다.
REPLY	읽을 때 인수가 주어 지지 않으면 설정된다.
SHELL	쉘은 호출될 때 이 이름에 대한 환경을 조사한다. 쉘은 등록가입할 때 설정되는 PATH, PS1, PS2, MAILCHECK, IFS, HOME, MAIL에 기정값을 준다.
TMOUT	탈퇴하기전에 입력을 대기하는 시간을 초로 지정한다.
VISUAL	emacs, gmacs 또는 vi와 같은 즉시지령편집을 위한 편집기를 지정한다.

설정된 변수들 현시 변수의 값을 인쇄하는 3개의 내부지령 set, env, typeset가 있다. set지령은 모든 변수들 즉 국부변수와 전역변수를 인쇄한다. env지령은 국부변수들만 인쇄한다. typeset지령은 모든 변수들, 옹근수들, 함수들 넘겨진 변수들을 인쇄한다. set -o지령은 Korn쉘을 의해 설정된 모든 추가선택들을 인쇄한다.

실례 10-36

```
1.  $ env                    #partial list
    LOGNAME=ellie
    TERMCAP=sun-cmd:te=\ E [ >4h:ti=\ 41:tc=sun:
    USER=ellie
    DISPLAY=:0.0
    SHELL=/bin/ksh
    HOME=/home/jody/ellie
    TERM=sun-cmd
    LD_LIBRARY_PATH=/usr/local/ow3/lib
    PWD=/home/jody/ellie/per1
2.  $    typeset
    export MANPATH
    export PATH
    interger    ERRNO
    export    FONTPATH
    interger   OPTIND
    function   LINENO
```

```
        export   OPENWINHOME
        export   LOGNAME
        function SECONDS
        interger   PPID
        PS3
        PS2
        export TERMCAP
        OPTARG
        export   USER
        export   DISPLAY
        function   RANDOM
        export    SHELL
        interger   TMOUT
        interger   MAILCHECK
3.    $   set
        DISPLAY=:0.0
        ERRNO=10
        FCEDIT=/bin/ed
        FMHOME=/usr/local/Frame-2.1x
        FONTPATH=/usr/local/ow3/lib/fonts
        HELPPATH=/usr/local/ow3/lib/locale: =/usr/local/ow3/lib/lelp
        HOME=/home/jody/ellie
        IFS=
        LD_LIBRARY_PATH=/usr/local/ow3/lib
        LINENO=1
        LOGNAME=ellie
        MAILCHECK=600
        MANPATH=/usr/local/ow3/share/man:/usr/local/ow3/man:/=/usr/local/man;
      =/usr/local/doctools/man:/usr/man
        OPTIND=1
        PATH=/home/jody/ellie/:/usr/local/ow3/bin:/usr/ucb:/usr/local/doctools/bin:/us
      r/bin:/usr/local:/usr/etc:/etc:/usr/spool/news/bin:/home/jody/ellie/bin:/usr/10
        PID=1332
        PSI=$
        PS2=＞
        PS3=3?
        PS4=+
        PWD=/home/jody/ellie/kshprog/joke
        RANDOM=4251
        SECOND=36
```

```
SHELL=/bin/ksh
TERM=sun-cmd
TERMCAP=sun-cmd:te=\ E [ >4h:ti=\ E [ >41:tc=sun:
     TMOUT=0
USER=ellie
_=pwd
name=Joe
Place=San Francisco
x=
```

4. set –0

```
allexport      off
bgnice         on
emace          off
gmacs          off
ignoreeof      off
interactive    on
keyword        off
markdirs       off
moniter        on
noexec         off
noclobber      off
noglob         off
nolog          off
nounset        off
privileged     off
restricted     off
trackall       off
verbose        off
viraw          off
xtrace         off
```

설명

1. env지령은 모든 환경(넘겨 진)변수들을 현시한다. 이 변수들은 습관상 대문자로 쓴다. 이것들은 자기가 작성한 프로쎄스로부터 자식프로쎄스에로 넘겨진다.
2. typeset지령은 모든 변수들과 그것들의 속성들, 함수들과 옹근수들을 현시한다. +추가선택이 있는 typeset지령은 변수들의 이름만을 현시한다.
3. set지령은 추가선택이 없으면 null값으로 설정된 변수들을 포함하여 설정된

모든 변수들 즉 국부변수와 넘겨 진 변수들을 인쇄한다.

4. -o추가선택이 있는 set지령은 on이나 off로 설정된 모든 내부지령변수들을 현시한다. 추가선택들을 취소하려면 더하기부호(+)를 사용하고 추가선택들을 설정하려면 덜기부호(-)를 사용한다. 실례로 set -o allexport는 allexport추가선택을 설정한다.

변수설정해제 국부변수나 환경변수들은 모두 unset지령을 사용하여 해제할수 있다(만일 변수들이 읽기전용함수로 설정되지 않으면).

실례 10-37

Unset name; unset TERM

설명

변수 name과 TERM은 이 쉘에 대하여 더이상 정의되지 않는다.

변수값인쇄 echo지령(Bourne와 C쉘에서 사용된)은 쉘에서도 사용할수 있지만 print지령이 더 많은 추가선택을 가지고 있고 더 효과적이다. echo지령과 print지령은 쉘에 내장되여 있다. print지령은 출력을 조종하는 많은 추가선택들을 가진다. 그것들을 표 10-10에서 보여 주었다.

표 10-10. print추가선택

추가선택	의 미
-r	탈출문자들을 보존한다.
-R	ksh가 -e나 -x를 print인수로 취급하지 않게 한다. 인수앞에 횡선이 있으면 그것을 없앤다(-n은 제외). \ t,\ c\ c는 특수한것으로 인식되지 않으며 \ 이 없이 인쇄될 때는 변화되지 않는다.
-un	출력을 서술자가 n인 파일로 방향바꾸기한다.
-n	출력에 행바꾸기가 없다. Echo -n과 같다.
-p	출력을 표준출력으로 보내지 않고 협동프로쎄스나 파이프(&|)로 보낸다.
-s	출력이 표준출력에로가 아니라 리력파일에 지령으로서 추가된다.
-	뒤에 오는 임의의 인수들은 print추가선택들이 아니다. 횡선은 련결부호가 있는 인수들을 허용한다. 실례: -2
-f	1988년이후 판본에서 printf를 대신하여 사용한다.

실례 10-38

1. $ print Hello my friend and neighbor!
   Hello my friend and neighbor!
2. $ print "Hello   friends"
   Hello   friends
3. $ print -r "\ n"

```
\ n
4.  $ print -s "date +%H"
    $ history –2
    132 print -s "date +%H"
    133 date +%H
    134 history -2

    $ r 133
    09
5.  $ print -n $HOME
    /home/jody/ellie
6.  $ var=world
    $ print ${var}wide
    worldwide
7.  $ print -x is an option
    ksh: print: bad option(s)
8.  $ print - -x is an option
    -x -is an option
```

설명

1. 쉘은 지령행을 구문해석하고 그것을 공백으로 분리되는 단어기호들로 나눈 다음 그 단어들을 인수로서 print지령에 넘긴다. 쉘은 단어사이에 있는 모든 공백들을 없앤다.
2. 인용부호는 한개문자렬을 만든다. 쉘은 문자렬을 한개 단어로 해석하고 공백을 보존한다.
3. 이것은 행추가선택이다. \ r와 같이 탈출문자들이 해석되지 않는다.
4. –s추가선택은 print지령의 인수들을 리력파일에 추가한다. 문자렬《date+%H》는 print지령에 대한 인수이다. Date문자렬이 리력목록에 지령으로 추가된 다음 r지령(리력의 redo지령)과 함께 실행된다.
5. –n추가선택은 행바꾸기문자를 없앤다. Korn쉘재촉문이 print지령의 출력과 같은 행에 있다.
6. 국부변수에 값 world가 대입된다. 대괄호는 변수들을 그에 추가된 문자들과 분리시킨다.
7. print함수에 대한 첫번째 인수가 횡선으로 시작될 때 print지령은 횡선앞에 아무것도 없으면 그 인수를 자기의 추가선택으로 해석한다.
8. 횡선은 추가선택으로 사용될 때 그것을 인쇄되는 문자렬의 첫번째 문자로 쓸수 있게 한다.

탈출문자렬 탈출문자들은 거꿀빗선이 앞에 붙은 한개 문자로 이루어 지는데 인용부호속에 있으면 특수한 의미를 가진다(표 10-11에서 보여 준다.).

print지령은 –r와 –R추가선택이 없으면 그다음에 오는 탈출문자들이 문자렬안에 놓일 때 출력을 형식화한다. 문자렬은 2중인용부호나 단일인용부호안에 넣어야 한다.

실례 10-39

1. **$ print '\ t\ t\ tHello\ n'**
   Hello
   $

2. **$ print "\ aTea XtTime! \ n\ n"**
   Ding (bell rings) Tea Time !

설명

1. 거꿀빗선들은 단일인용부호나 2중인용부호속에 넣어야 한다. \ t탈출문자들은 타브를 현시하고 \ n은 행바꾸기를 현시한다. 출력은 처음에 3개의 타브, 그다음에 문자렬 Hello, 마지막에 행바꾸기로 된다.
2. \ a탈출문자들은 종이 울리게 하고 (\ 07)\ t는 한개 타브를 만든다. 2개의 \ n문자렬은 2개의 행바꾸기를 인쇄한다.

표 10-11. 조종문자렬

거꿀빗선	의 미
\ a	종울림문자
\ b	후퇴(뒤걸음문자)
\ c	행바꾸기를 없애고 \ c다음에 오는 임의의 인수들을 무시한다.
\ f	폐지바꾸기
\ n	행바꾸기
\ r	귀환
\ t	타브
\ v	수직타브
\ \	거꿀빗선
\ Ox print \	0124에서와 같이 1, 2 또는 3자리 ASCII값을 가지는 8 bit문자
\ E	1988년이후판본에서만 있다. 조종문자렬로 사용된다.

변수식과 확장변경자 변수식은 특수변경자들을 사용하여 검사하거나 수정할수 있다. 변경자는 지름조건검사를 보장하여 변수가 설정되였는가를 검사하고 그다음에 변경자에 따라 변수의 기정값을 대입할수 있다. 이 식들은 if와 elif와같은 조건명령문들에서 사용할수 있다(표 10-12에서 보여 준다.). 옹근두점을 변경자와 함께 사용하지 말아야 한다. 옹근두점을 변경자와 함께 사용하면 변수가 설정되지 않았는가 즉 빈값인가를 검사한다. 옹근두점이 없으면 빈값으로 설정된 변수를 설정된 변수로 본다.

표 10-12. 변수식과 변경자

식	기 능
${variable:-word}	만일 변수가 설정되고 빈값이 아니면 그의 값을 치환한다. 그렇지 않으면 word를 치환한다.
${variable:=word}	변수가 설정되지 않으면 또는 빈값이면 그것을 word로 설정한다. 변수의 값은 영구적으로 치환한다. 위치파라메터들은 이런 식으로 대입될수 없다.
${variable:+word}	변수가 설정되여 빈값이 아니면 word를 치환한다. 그렇지 않으면 아무것도 치환하지 않는다.
${variable:?word}	변수가 설정되여 빈값이 아니면 그 값을 치환한다. 그렇지 않으면 word를 인쇄하고 쉘에서 탈퇴한다. 만일 word가 생략되면 통보문《 parameter null or not set》가 인쇄된다.

실례 10-40

(Using Temporary Default Values)

1. $ **fruit=peach**

2. $ **print ${fruits:-plum}**
   peach

3. $ **print ${newfruit:-apple}**
   apple

4. $ **print $newfruit**

5. $ **print ${TERM:-vtl20}**
   sun-cmd

설명

1. 변수 furint에 값 peach가 대입된다.
2. 특수변경자는 변수 fruit가 설정되였는가를 검사한다. 설정되였으면 값 peach가, 설정되지 않았으면 값plum이 인쇄된다.
3. 변수 newfruit가 설정되지 않는다. 값 apple가 인쇄된다.
4. 변수 newfruit가 설정되지 않는다. 따라서 아무것도 인쇄되지 않는다.단계 3에형식은 단어 apple로 치환되여 인쇄된다.
5. TERM변수가 설정되지 않았으므로 기정값 vt120이 현시된다. 이 실례에서 말단은 이미 sun-cmd, 전문작업용콤퓨터로 설정되여 있다.

실례 10-41

(Assigning Permanent Default Values)

1. $ **name=**
2. $ print ${name:=Pattiy}

Patty
3. $ print $name
Patty
4. $ print ${TERM:=vt120}
vt12o
$ print $TERM
vtl20

설명

1. 변수 name에 null값이 설정된다.
2. 특수변경자:=는 변수 name이 빈값이 아닌 어떤값으로 설정되였는가를 검사한다. 만일 설정되였다면 그것은 변화되지 않는다. 만일 그것이 빈값이거나 설정되지 않았으면 갈기부호오른쪽에 있는 값으로 대입된다. name변수가 빈값으로 설정되였으므로 patty가 name에 대입된다. 설정은 영구적이다.
3. 변수 name은 여전히 값 patty를 가진다.
4. 만일 변수 TERM이 설정되지 않으면 기정값 vt120이 영구적으로 대입된다.

실례 10-42

(Assigning Temporary Alternate Value)
1. $ foo=grapes
2. $ print ${foo:+pears}
pears
$ print $foo
grapes

설명

1. 변수 foo에 값 grapes가 대입된다.
2. 특수변경자 :+는 변수가 설정되였는가를 검사한다. 만일 그것이 설정되여 있으면 림시로 grapes로 재설정된다. 설정되지 않았으면 빈값으로 되돌려 준다.

실례 10-43

(Creating Error Messages Based on Default Values)
1. $ **print ${namex:?"namex is undefined"}**
ksh: namex is undefined
2. $ **print ${y?}**
ksh: y: parameter null or not set

설명

1. :?변경자는 변수가 설정되였는가를 검사한다. 설정되지 않았으면 ?의 오른쪽에 있는 문자렬이 그 변수의 이름뒤에 붙어 표준출력으로 인쇄된다.
2. ?다음에 통보가 없으면 Korn쉘은 기정통보를 표준오유에로 보낸다. 옹근두

점이 없으면 ?변경자는 빈값으로 설정된 변수를 설정된 변수로 보고 통보문을 인쇄하지 않는다.

실례 10-44

(Line from a System Script)
if ["${uid:=0}" –ne 0]

설명

UID(사용자 ID)가 값을 가지면 변화되지 않는다. 만일 그것이 값을 가지지 않으면 값0(주사용자)이 대입된다. 변수의 값이 0이 아닌가 검사된다. 이 행은 /etc/shutdown프로그람(SVR4/Solaris2.5)으로부터 얻어 졌다. 여기서는 변수변경자를 어떻게 사용하는가를 보여 주었다.

부분문자렬의 변수확장 인수들을 대신하는 패턴은 문자렬의 앞이나 끝으로부터 어떤 부분을 떼내는데 사용된다. 이 연산자들을 자주 사용하여 경로의 머리부나 꼬리부로부터 경로이름요소들을 없앤다. 표 10-13에서 보여 준다.

표 10-13. 변수확장부분문자렬

식	기 능
${variable%pattern}	변수값의 제일 작은 꼬리부분을 패턴과 비교하여 그것을 지우기한다.
${variable%%pattern}	변수값의 제일 큰 꼬리부분을 패턴과 비교하여 그것을 지우기한다.
${variable#pattern}	변수값의 제일 작은 시작부분을 패턴과 비교하여 그것을 지우기한다.
${variable##pattern}	변수의 제일 큰 시작부분을 패턴과 비교하여 그것을 지우기한다.

실례 10-45

1. $ **pathnames"/usr/bin/local/bin"**
2. $ **print ${pathnames/bin*}**
   /usr /bin/local

설명

1. 국부변수 pathname에 /usr/bin/local/bin이 대입된다.
2. %는 패턴 /bin과 그뒤에 오는 임의의 물음표들을 포함하는 경로이름의 제일 작은 꼬리부분을 지우기한다. 즉 /bin을 없앤다.

실례 10-46

1. $ **pathname="usr/bin/local/bin"**

2. $ **print ${pathname%%/bin*}**

   /usr

설명

1. 국부변수 pathname에 /usr/bin/loclal/bin이 대입된다.
2. 패턴 /bin과 그뒤에 오는 임의의 수물음표들을 포함하는 경로이름의 제일 큰 꼬리부분을 지우기한다. 즉 /bin/local/bin을 없앤다.

실례 10-47

1. $ **pathname=/home/lilliput/jake/.cshrc**
2. $ **print ${pathname#/home}**

   /1illiput/jake/.cshrc

설명

1. 국부변수 pathname에 /home/liliput/jake/.cshrc가 대입된다.
2. #는 패턴 /home을 포함하는 경로이름의 제일 작은 시작부분을 지우기한다. 즉 /home이 경로변수의 시작점에서 없어 진다.

실례 10-48

1. $ **pathname=/home/liliput/jake/.cshrc**
2. $ **print ${pathname##*/}**

   .cshrc

설명

1. 국부변수 pathname에 /home/liliput/jake/.cshrc가 대입된다.
2. ##는 마지막에 빗선이 오고 그 앞에 임의의 물음표가 놓이는 경로이름의 제일 큰 시작부분을 지우기한다. 즉 경로변수로부터 /home/liliput/jake를 없앤다.

변수속성: typeset지령 typeset지령을 사용하여 변수의 선택(소문자인가, 대문자인가), 너비 그리고 왼쪽 또는 오른쪽 정돈과 같은 변수의 속성들을 조종할수 있다. typeset지령이 변수의 속성을 변화시키면 그 변화는 영구적이다. typeset함수는 여러가지 기능들을 가진다. 표 10-14에서 보여 준다.

실례 10-49

1. $ typeset –u name=”john doe”

```
$ print "$name"
JOHN DOE
# Changes all characters to uppercase.
```

2. $ typeset –1 name

```
$ print $name
john doe                    # Changes all characters to lowercase.
```

3. $ typeset –L4 name

```
$ print $name
john                        # Left-justified fixed-width 4-character field.
```

4. $ typeset –R2 name

```
$ print $name         # Right-justified fixed-width 2-character field.
hn
```

5. $ name="John Doe"

```
$ typeset –z15 name          # null-padded sting, 15-space field width.
width
$ print "$name"
John Doe
```

6. $ typeset –LZ15 name   # Left-justified, 15-space field width.

```
$ print "$name$name"
John Doe          John Doe
```

7. $ integer n=25

```
$ typeset –Z15 n                 # Left-justified, zero-padded integer.
$ print "$n"
00000000000025
```

8. $ typeset –lL1 answer=Yes   # Left-justify one lowercase letter.

```
$ print $answer
y
```

설명

1. typeset지령에 대한 –u추가선택은 변수의 모든 문자들을 대문자로 변화시킨다.
2. typeset지령에 대한 –l추가선택은 변수의 모든 문자들을 소문자로 변화시킨다.
3. typeset지령에 대한 –L추가선택은 변수이름을 왼쪽으로 정돈하고 4개 문자로 된 문자렬 john으로 변화시킨다.
4. typeset지령에 대한 –R추가선택은 변수이름을 오른쪽으로 정돈되고 2개문자로 된 문자렬 hn으로 변화시킨다.
5. 변수 name이 John Doe로 설정된다. typeset지령에 대한 –Z추가선택은 문

자렬을 변수값으로 보충된 15자리 문자렬로 전환시킨다. 공백을 보존하기 위해 $를 인용부호안에 넣는다.

6. 변수 name이 왼쪽 정돈되고 빈값이 보충된 15자리 문자렬로 전환된다
7. 변수 n은 값 25로 대입된 옹근수(표 10-14에서 typeset -i를 보여 준다.)이다. typeset지령은 옹근수 n을 0으로 채워 진 15자리 왼쪽 정돈된 수로 전환한다.
8. 변수 answer에 값 Yes가 대입되고 왼쪽 정돈되고 한개의 소문자로 이루어 진 문자렬로 전환된다(이것은 스크립트에서 사용자입력을 처리할 때 아주 쓸모 있다.).

표 10-14. typeset지령의 기타사용

지 령	기 능
typeset	모든 변수들을 현시한다.
typeset-i num	num에 대한 옹근수값들만을 받는다.
typeset-x	반출된 변수들을 현시한다.
typeset abc	함수안에서 정의되면 a, b, c를 국부변수로 작성한다.
typeset-r x=foo	x를 foo로 설정한 다음에 그것을 읽기전용으로 한다.

위치파라메터 위치파라메터라고 하는 특수내부변수들은 지령행으로부터 인수들을 넘길 때 또는 함수들에 사용되여 그 함수에 넘겨 진 인수들의 값을 유지할 때 쉘스크립트들에서 사용된다. 이 변수들은 파라메터목록에서 각자의 위치를 현시하는 수 1, 2, 3 등에 의해 참조되기때문에 위치파라메터라고 부른다. 표 10-15에서 보여 준다. 쉘스크립트의 이름은 $0변수에 기억된다. 위치파라메터들은 set지령으로 설정되고 재설정될수 있다.

표 10-15. 위치파라메터

식	기 능
$0	현재쉘스크립트의 이름을 참조한다.
$1-$9	위치파라메터 1부터 9까지
${10}	위치파라메터 10
$#	위치파라메터들의 개수를 표시한다.
$*	모든 위치파라메터들을 표시한다.
$@	2중인용부호안에 있을 때를 제외하고 $*와 같다.
"$*"	"$1 $2 $3" 등을 표시한다.
"$@"	"$1" "$2" "$3" 등을 표시한다.

실례 10-50

```
1. $ set tim bill ann fred
   $ print $*          # Prints all the positional parameters.        tim
```

```
    bill    ann fred
2.  $ print $1               # Prints the first position.
    tim
3.  $ print $2 $3            # Prints the second and third position.
    bill ann
4.  $ print $#               # Prints the total number of positional
   4                         # parameters.
5.  $ set a b c d e f g h i j k l m
    $ print $10              # Prints the first positional parameter
    a0                       # followed by a 0.
    $ print ${10) ${11}   # Prints the 10th and. llth positions.
    j k
6.  $ print $#
    13
7.  $ print $*
    a b c d e f g h i f j k l m
8.  $ set file1 file2 file3
    $ print \ $$#
    $3
9.  $ eval print \ $$#
    file3
```

설명

1. set 지령은 값들을 위치파라메터들에 대입한다. 특수변수 $*는 모든 파라메터모임을 포함한다.
2. 첫번째 위치파라메터의 값 tim이 현시된다.
3. 두번째와 세번째 파라메터들의 값 bill과 ann이 현시된다.
4. 특수변수 $#는 현재 설정된 위치파라메터들의 개수를 가지고 있다.
5. set지령은 모든 위치파라메터들을 재설정한다. 초기파라메터목록이 지워 진다. 9보다 큰 임의의 위치파라메터들을 인쇄하려면 대괄호를 리용하여 2개의 수자를 함께 괄호속에 넣는다. 그렇지 않으면 첫번째 위치파라메터의 값이 인쇄되고 그다음에 그것의 번호가 인쇄된다.
6. 위치파라메터들의 개수는 현재 13이다.
7. 모든 위치파라메터들의 값이 인쇄된다.
8. 표식 $가 탈퇴된다. $#는 인수들의 개수이다. print지령은 위치파라메터들의 개수가 뒤에 붙은 $3을 현시한다.
9. eval 지령은 지령을 실행하기전에 지령행을 다시 해석한다. 쉘에 의해 첫번째로 해석될 때 print는 $3, 두번째 해석때 eval다음에 있는 print는 $3의 값인 file3을 현시한다

13. 기타특수변수

Korn쉘은 표 10-16에서 보여 준것과 같은 몇개의 특수내부변수들을 가진다.

표 10-16. 특수변수

변 수	의 미
$$	쉘의 PID
$-	현재 설정된 ksh추가선택들
$?	마지막으로 실행된 지령의 탈퇴값
$!	마지막으로 배경에 넣어 진 과제의 PID

실례 10-51

1. $ **print The pid of this shell is $$**
   The pid of this shell is

2. $ **print The options for this korn shell are $-**
   The options for this korn shell are Ismh

3. $ **grep dodo /etc/passwd**
   $ **print $?**
4. $ **sleep 25&**
   [1] 400
   $ **print $!**
   400

설명

1. $$변수는 프로쎄스에 대한 PID값을 가진다.
2. $-변수는 대화형 Korn쉘에 대한 모든 추가선택들을 현시한다.
3. grep지령은 /etc/passwd파일에서 문자렬 dodo를 탐색한다. 변수는 마지막으로 실행된 지령의 탈퇴상태를 가진다. grep로부터 되돌려 진 값이 1이므로 grep는 탐색에서 실패했다고 본다. 0탈퇴상태가 성공적인 탈퇴를 현시한다.
4. sleep지령에 붙은 &는 배경에서 지령이 실행되게 한다. $!변수는 배경의 마지막에 놓여 진 지령의 PID번호를 가진다.

14. 인용부호

인용부호를 사용하여 특수메타문자들이 해석되지 않게 한다. 인용부호들은 모든 쉘 스크립트들에서 오유수정할 기본적인 혼란을 일으킬수 있다. 단일인용부호들은 반드시 대응되여야 한다. 이것들은 특수메타문자들이 쉘에 의해서 해석되지 않도록 한다. 2중인용부호들도 대응되여야 한다. 이것들도 대부분의 메타문자들이 쉘에 의해서 해석되지 않

게 하지만 변수와 지령바꿔넣기문자들이 처리되게 한다. 단일인용부호들은 2중인용부호들을 보호하고 2중인용부호들은 단일인용부호들을 보호한다. Bourne쉘과 달리 Korn쉘은 인용부호가 대응되지 않았다면 그 행과 함께 표준오유에로 오유통보를 보내여 어디서 인용부호가 대응되지 않았는가를 알려 준다.

거꿀빗선 거꿀빗선을 사용하여 한개 문자가 해석되지 않게 보호한다(탈퇴시킨다.).

실례 10-52

1. $ **print Where are you going\ ?**
   Where are you going?
2. $ **print Start on this line and**
   > **go to the next line.**
   Start on this line and go to the next line.

설명

1. 특수메타문자 ?는 거꿀빗선으로 탈퇴된다. 쉘은 파일이름확장을 위해 이 문자를 해석하지 않는다.
2. 행바꾸기문자가 탈퇴된다. 다음행이 첫번째 행의 부분으로 된다. >은 Korn쉘의 2차재촉문이다.

단일인용부호 단일인용부호는 대응되여야 한다. 이것들은 모든 메타문자들을 해석으로부터 보호한다. 단일인용부호를 인쇄하려면 그것을 2중인용부호속에 넣거나 거꿀빗선으로 탈퇴시켜야 한다.

실례 10-53

1. $ print 'hi there
   > how are you?
   When will this end?
   > When the quote is matched
   > oh'
   hi there
   how are you?
   When will this end?
   When the quote is matched
   Oh
2. $ print 'Do you need $5.00?'
   Do you need $5.00?
3. $ print 'Mother yelled, "Time to eat!"'
   Mother yelled, "Time to eat!"

설명

1. 단일인용부호는 행에서 대응되지 않는다. Korn쉘은 2차재촉문을 현시한다. 단일인용부호가 대응되기를 기다린다.
2. 단일인용부호는 모든 메타문자들을 해석으로부터 보호한다. 이 실례에서 $?는 쉘로부터 보호되여 문자로 취급된다.
3. 단일인용부호는 이 문자렬에서 2중인용부호를 보호한다. 여기서 2중인용부호는 회화체의 인용부호이다.

2중인용부호 2중인용부호는 대응되여야 하고 변수와 지령바꿔넣기를 허용하며 임의의 다른 특수문자들이 쉘에 의해서 해석되는것을 보호한다.[4]

실례 10-54

1. $ **name=Jody**
2. $ **print "Hi $name, I'm glad to meet you!"**
   Hi Jody, I'm glad to meet you!

3. $ **print "Hey $name, the time is `date`"**
   Hey Jody, the time is Fri Dec 18 14:04:11 PST 2001

설명

1. 변수 name에 문자렬 Jody가 대입된다.
2. 문자렬을 둘러 싼 2중인용부호는 모든 특수메타문자들을 해석으로부터 보호하는데 $name에서 $만은 례외이다. 변수바꿔넣기가 2중인용부호속에서 진행된다.
3. 변수바꿔넣기와 지령바꿔넣기가 둘다. 2중인용부호안에 넣어 질 때 진행된다. 변수 name이 확장되고 거꿀인용부호안에 있는 지령 date가 실행된다.

15. 지령바꿔넣기

지령바꿔넣기는 한개 지령의 출력을 변수에 대입하거나 문자렬에 대입할 때 사용된다. Bourne쉘과 C쉘은 거꿀인용부호를 사용하여 지령바꿔넣기를 진행한다. Korn셸은 거꿀인용부호를 쓰지않고 지령을 괄호안에 넣어서 처리한다. 왜냐하면 이렇게 하여야 괄호규칙이 더 간단하고 지령들을 더 쉽게 겹쳐 진 지령들은 만들수 있다.

4. 지령바꿔넣기를 위해 거꿀인용부호를 사용하는것은 Bourne쉘과 C쉘에서 사용하던 낡은 방식이다. 맞춤법은 정확하지만 Korn쉘은 이 절에서 새로운 방법을 설명한다.

형식

`Unix 지령` *# Old method with backquotes*
$(Unix 지령) *# New method*

실례 10-55

(Old Way)

1. $ **print "The hour is 'date +%H'"**
   The hour is 09
2. $ **name='nawk –F: "{print $1}' database'**
   $ **print $name**
   Ebenezer Scrooge
3. $ **ls 'ls /etc'**
   shutdown
4. $ **set 'date'**
5. $ **print $*"**
   Sat Oct 13 09:35:21 PDT 2001
6. $ **print $2 $**
   Oct 2001

설명

1. date지령의 출력이 문자렬에 치환된다.
2. nawk지령의 출력이 변수 name에 대입되여 현시된다.
3. 거꿀인용부호안에 있는 ls지령의 출력은 /etc등록부의 파일들의 목록이다. 파일이름들은 첫번째 ls지령에 대한 인수들이다. 현재등록부에 있는것과 같은 이름을 가진 /etc에 있는 모든 파일들이 현시된다.
4. set지령은 date지령의 출력을 위치파라메터들에 대입한다. 공백은 단어들의 목록을 개개의 파라메터들로 분리한다.
5. $*변수는 모든 파라메터들을 가진다. date지령의 출력은 $*변수에 기억된다. 매 파라메터는 공백에 의해 분리된다.
6. 두번째와 여섯번째 파라메터들이 인쇄된다. ksh지령바꿔넣기에서 거꿀인용부호대신에 사용할수 있는 다른 방법을 실례 10-56에 보여 주었다.

실례 10-56

(The New *ksh* Way)

```
1. $ d=$(date)
   print $d
   Sat Oct 20 09:35:21 PDT 2001
2. $ line = $(< filex)
3. $ print The time is $(date +%H)
   The time is 09
4. $ machine=$(uname –n)
   $ print $machine
   jody
5. $ dirname=”$(basename $(pwd)) “        # Nesting commands
   $ print $dirname
   bin
```

설명

1. date지령이 괄호로에 닫겨 있다. 지령의 출력은 식으로 복귀되여 변수뒤에 대입되고 현시된다.
2. 파일의 입력이 변수 line에 대입된다. <filex표기는 `at filex`와 같은 효과를 가진다. 지령바꿔넣기는 괄호앞에 표식 $가 있을 때 괄호안에서 진행된다.
3. UNIX의 date지령과 그의 시간인수 +%H가 괄호안에 닫겨 진다. 지령바꿔넣기가 진행되고 그 결과가 인쇄문자렬안에 놓인다.
4. 지령바꿔넣기가 진행되였다. uNIX uname지령의 출력이 변수 machine에 대입된다.
5. 변수 dirname을 현재작업등록부의 이름으로 설정하려면 지령바꿔넣기가 겹쳐져 있다. pwd지령이 먼저 실행되여 현재작업등록부의 완전경로이름을 UNIX지령 basename에 인수로서 넘긴다. basename지령은 경로이름의 마지막요소를 제외한 모든것을 없애 버린다. 거꿀인용부호안에서는 지령들을 겹쳐 쓸수 없다.

16. 함수

이 절에서는 함수들을 설명하는데 그것들을 대화형으로 사용하거나 초기화파일에 기억시킬수 있다. 앞으로 스크립트들에 대해 언급할 때 함수들에 대하여 더 깊이 학습하기로 한다. 별명이 명백하지 않을 때 즉 인수를 넘겨야 할 때 함수들을 사용할수 있다. 함수들은 보통 사용자초기화파일 .profile에서 정의된다. 이것들은 작은 스크립트와 비슷하지만 스크립트와 달리 함수들은 현재환경에서 실행된다. 즉 쉘은 함수를 실행시키기 위해 자식프로쎄스를 파생시키지 않는다. 그 함수를 호출한 쉘에서 모든 변수들이 공유된다. 보통 함수들을 사용하여 스크립트의 모듈성을 개선한다. 함수들은 일단 정의되면 반

복하여 사용할수 있고 다른 등록부에 기억시킬수 있다. 함수들은 호출되기전에 정의되여야 한다. 이것들을 정의하는데 2가지 형식이 있다.

한 형식은 Bourne쉘로부터 넘어 온것이고 다른것은 Korn쉘에서 새롭게 도입된것이다. 함수들은 쉘의 한 호출로부터 다음호출에로 넘겨 질수 있다. typeset함수와 unset 지령을 사용하여 함수들을 현시하고 해제한다. 표 10-17에서 보여 준다.

표 10-17. typeset지령과 함수

지 령	기 능
typeset -f	함수들과 그 정의들을 현시한다. 함수들은 typeset-f에 대한 별명이다.
typeset +f	함수이름들만을 현시한다.
unset -fname	함수를 해제한다.

함수정의 함수들을 정의하는데는 2가지형식이 있는데 그것은 Bourn쉘형식(웃준위호환성을 위해 허용된)과 새로운 Korn쉘형식이다. 함수는 사용하기전에 정의하여야 한다.

형식

(Bourne Shell)
함수이름 (){ 지령 ; 지령;}[5]

(Korn Shell)
function 함수이름 { 지령; 지령;}

실례 10-57

```
1.  $ function fun < pwd; ls; date; }

2.  $ fun
    /home/jocly/ellie/prac
    abc       abcl23    file1.bak   none        nothing tmp
    abcl      abc2      file2       nonsense    nowhere touch
    abcl22    file1     file2.bak   noone       one
    Tue Feb 9 11:15:48 PST 2001

3.  $ function greet { print "Hi $1 and $2"; }

4.  $ greet tom joe           # Here $1 is torn and. $2 is joe

5.  $ set jane nina lizzy
```

5. POSIX 표준은 Bourne쉘문법으로 함수들을 정의하지만 변수와 트랩프들은 새로운 Korn쉘정의에서와 같이 그 유효범위가 국부적이 아니다.

6. $ print $*
   jane nina lizzy
7. $ greet tom joe
   Hi tom and. joe
8. $ print $1 $2
   jane nina

설명

1. 함수 fun이 이름 지어 지고 정의된다. 이름뒤에 대괄호안에 닫겨 진 지령들의 목록이 놓인다. 매 지령은 반두점에 의해 분리된다. 첫번째 대괄호뒤에 공백이 있어야 한다. 그렇지 않으면 ksh:syntax error:`}'unexpected와 같은 문법오유통보가 나타난다. 함수는 사용되기전에 정의되여야 한다.
2. 함수는 호출될 때 스크립트나 별명과 같이 실행된다. 함수정의안에 있는 매 지령이 차례로 실행된다.
3. greet함수에서 2개의 위치파라메터들이 사용된다. 함수에 인수가 주어 질 때 위차파라메터들이 그 값들로 대입된다.
4. 함수 tom과 joe에 대한 인수들이 $1과 $2에 각각 대입된다. 함수에 있는 위치파라메터들은 그 함수에 대해서만 사용할수 있고 함수밖에서는 사용되지 않는다.
5. 위치파라메터들은 지령행에서 설정된다. 이 값들은 함수에서 설정된것들과 아무 련관도 없다.
6. $*는 현재 설정된 위치파라메터들의 값들을 표시한다.
7. 함수 greet가 호출된다. 위치파라메터 $1과 $2에 대입된 값들은 각각 tom과 joe이다.
8. 지령행에서 대입된 위치변수들은 함수에서 설정된것들에 의해 영향을 받지 않는다.

함수와 별명 지령행을 처리할 때 쉘은 특수내부지령들전에 별명들을 찾고 그다음 특수내부지령들을 찾고 마지막에 함수들을 찾는다. 만일 함수가 내부지령과 같은 이름을 가지고 있다면 내부지령이 함수보다 더 높은 우선권을 가진다. 특수한 내부지령에 대한 별명을 정의할수 있으며 이때 함수이름은 그 별명이름으로 하여 처리순서를 무시할수 있다.

실례 10-58

(The ENV File)

1. alias cd=_cd
2. function _cd {
3. \ cd $1

4. print ${basename $PWD}

```
5.  }

(The Command Line)
 $ cd /
/
$ cd $HOME/bin
bin
$ cd ..
ellie
```

설명

1. cd에 대한 별명이 _cd로 대입된다.
2. 함수 _cd가 정의된다. 열린 대괄호는 함수정의의 시작을 현시한다.
3. 별명앞에 거꿀빗선이 오면 별명바꿔넣기가 진행되지 않는다. cd앞에 붙은 거꿀빗선은 별명이 아니라 내부지령 cd지령을 실행시킨다. 거꿀빗선이 없으면 함수는 재귀적이고 쉘은 오유통보 cd_:recursion too deep를 현시한다. $1는 cd에 넘겨 지는 인수(등록부의 이름)이다.
4. 등록부의 이름(완전경로이름이 아닌)이 인쇄된다.
5. 닫는 대괄호는 함수정의의 끝을 표시한다.

함수현시 함수와 그 정의를 현시하기 위해 typeset지령을 사용한다.

실례 10-59

```
(The Command Line)
1.  $ typeset -f
    function fun
    {
    pwd; ls; date; }
    function greet
    {
    print "hi $1 and $2"; }

2.  $ typeset +f
    fun
    greet
```

설명

1. -f추가선택이 설정된 typeset지령은 함수들과 그 정의를 현시한다.
2. +f추가선택이 설정된 typeset지령은 정의된 함수들의 이름만을 현시한다.

함수해제 함수가 해제될 때 그것은 쉘기억기로부터 지우기된다.

실례 10-60

```
(The Command Line)
1.  $ typeset -f
     function fun
     {
          pwd; ls; date; }
     function greet
     {
     print “hi $1 and $2”; }
2.  $ unset –f fun
3.  $ typeset -f
     function greet
     {
 print “hi $1 and $2”; }
```

설명

1. typeset -f지령은 함수와 그 정의를 현시한다. 2개의 함수들 fun과 greet가 현시된다.
2. -f추가선택을 가진 내부지령 unset는 fun함수정의를 해제하고 그것을 쉘기억기로부터 지우기한다.
3. fun함수는 typeset -f지령이 실행될 때 정의된 함수로 더이상 나타나지 않는다.

17. 표준입출력과 방향바꾸기

쉘은 프로그람이 시작될 때마다 3개의 파일(흐름이라고 한다.) stdin, stdout, stderr을 연다. 표준입력은 보통 건반으로부터 오며 그 파일형식자는 0이다. 표준출력은 보통 화면으로 나가며 파일형식자는 1이다. 표준오유는 보통 화면으로 나가며 파일형식자는 2이다. 표준입력과 출력 그리고 오유는 파일에 방향바꾸기되거나 파일로부터 방향바꾸기될수 있다. 표 10-18에서 방향바꾸기연산자들에서 보여 준다.

표 10-18. 방향바꾸기

연산자	기 능
<	입력을 방향바꾸기한다.
>	출력을 방향바꾸기한다.
>>	출력을 추가한다.
2>	오유를 방향바꾸기한다.
1<&2	출력을 오유가 나가는곳으로 방향바꾸기한다.
2>&1	오유를 출력이 나타나는곳으로 방향바꾸기한다.

실례 10-61

```
(The Command Line)
1.  $ tr '[A-Z]' '[a-z]' < myfile     # Redirect input
2.  $ ls > lsfile                   # Redirect output
    $ cat lsfile
    dir1
    dir2
    file3
3.  $ date >> lsfile                # Redirect and append output
    $ cat lsfile
    dir1
    dir2
    file1
    file3
    Mon Sept 17 12:57:22 PDT 2001
4.  $ cc prog.c 2> errfile          # Redirect error
5.  $ find . –name\ *.c –print > founditfile 2> /dev/null
6.  $ find . –name\ *.c –print > foundit 2>&1
7.  $ print "File needs an argument" 1>&2
8.  $ function usage { print "Usage: $0 [-y] [-g] filename" 1>&2;
    exit 1; }
```

설명

1. 표준입력은 파일 myfile로부터 UNIX의 tr지령에로 방향바꾸기된다. 모든 대문자들이 소문자들로 변환된다.
2. ls지령은 출력을 파일 lsfile에로 방향바꾸기한다.
3. date지령의 출력은 lsfile에로 방향바꾸기되여 추가된다.
4. 파일 prog.c가 번역된다. 만일 번역이 실패하면 표준오유가 errfile에로 방향바꾸기된다
5. find지령은 현재작업등록부에서 .c로 끝나는 파일이름탐색을 시작하고 파일들을 `founditfile`파일에로 인쇄한다. find지령으로부터 오유가 /dev/null에 보내진다.
6. find지령은 현재작업등록부에서 .c로 끝나는 파일이름들을 탐색하기 시작하고 파일들을 foundit파일에로 인쇄한다. 표준오유(파일형식자2)는 표준출력(파일서술자1)이 보내지게 되는 foundit파일에로 보내 지게 된다.
7. print지령은 자기의 통보문을 표준오유에로 보낸다. 표준출력은 표준오유와 합쳐 진다. 즉 표준출력은 표준오유가 나가는 말단으로 방향바꾸기된다. 이

렇게 하여 《좋은》 출력으로부터 오유통보들을 분리시킬수 있다.
8. 함수 usage가 정의된다. 이 함수는 호출될 때 사용법통보를 인쇄하고 출력을 표준오유로 보낸 다음 탈퇴한다. 이러한 형태의 함수는 스크립트에서 자주 사용된다.

exec지령과 방향바꾸기 exec지령을 사용하여 현재프로그람을 실행될 프로그람으로 교체할수 있다.

또한 exec지령을 사용하여 자식쉘을 작성하지 않고도 표준출력이나 표준입력을 변화시킨다. 만일 파일이 exec지령으로 열린다면 그다음에 오는 read지령이 파일지적자를 파일끝까지 한번에 한행씩 파일아래로 이동시킨다. 파일을 처음부터 다시 읽자면 그 파일을 닫아야 한다.

그러나 이때 cat와 sort와 같은 UNIX편의프로그람을 사용하면 조작체계는 매 지령이 완성된 다음에 파일을 닫는다.

표 10-19에서 exec기능에서 보여 준다.

표 10-19. exec지령

지 령	기 능
exec ls	ls는 쉘이 있는 곳에서 실행된다. ls가 끝나면 그것이 시작되였던 셸이 복귀하지 않는다.
exec<filea	표준입력을 읽기 위해 filea를 연다.
exec>filex	표준출력을 쓰기 위해 filex를 연다.
exec 2>errors	표준오유를 쓰기 위해 errors를 연다.
exec2>/dev/console	모든 오유통보들을 조종탁으로 보낸다.
exec3<datfile	입력을 읽기 위해 datfile을 파일서술자 3으로 연다.
Sort<&3	datfile이 분류된다.
exec4>newfile	쓰기 위해 newfile을 파일서술자 4로 연다.
ls>&4	ls의 출력이 newfile에로 방향바꾸기된다.
Exec5<&4	fd5를 fd4로 복사한다. 2개의 서술자들은 newfile을 표시한다.
Exec3<&-	파일형식자 3인 datfile을 닫는다.

18. 방향바꾸기와 자식쉘

지령의 출력이 화면으로부터 파일에로 방향바꾸기될 때 Korn쉘은 그림 10-2와 같이 자식쉘을 작성하여 파일형식자들을 다시 배렬한다.

그림 10-2. 표준출력과 오유의 방향바꾸기

19. 파이프

파이프는 파이프기호의 왼쪽에 있는 지령의 출력을 받아서 그것을 파이프기호의 오른쪽에 있는 지령의 입력으로 보낸다. 흐름선은 한개이상의 파이프로 이루어 질수 있다.

실례 10-62

1. $ **who > tmp**
2. $ **wc –l tmp**
   4 tmp
3. $ **rm tmp**
4. $ **who | wc –l** *# Using the pipe*

설명

행 1부터 3까지의 목적은 등록가입한 사람들의 수를 계산하고 지령의 출력을 파일(tmp)에 기억하고 wc-l을 사용하여 tmp파일에 있는 행들의 수를 계산한 다음에 tmp파일을 지우기하는것이다. 즉 등록가입한 사람들의 수를 구한다. 파이프는 한개지령에서 같은 과제를 수행한다.

1. who지령의 출력이 tmp파일에로 방향바꾸기된다.
2. wc-l지령이 tmp에 있는 행들의 수를 현시한다.
3. tmp파일이 지우기된다.
4. 파이프를 사용하여 우의 3단계 1, 2, 3을 한단계로 진행할수 있다. who지령의 출력은 닉명의 핵심부완충기(디스크공간을 요구하는 림시파일대신)에로 보내진다. wc-l지령은 완충기로부터 그것을 읽어 들여 출력을 화면으로 보낸다.

실례 10-63

1. $ ls | more
   < lists (ls) all files one page at a time (more) >
2. $ du ~ | sort –n | sed –n '$p'
   72388 /home/jody/ellie
3. $ cat | lp or cat | lpr

설명

1. ls출력은 입력을 받는 more지령에로 파이프된다. 출력이 한번에 한페지씩 현시된다.
2. du지령의 출력(디스크사용법)이 수자로 분류되고 sed지령(흐름편집기)에 파이프되며 sed지령은 마지막행($P)만을 현시한다.
3. cat지령은 표준입력으로부터 읽어 들인다. 출력이 행인쇄기(SUR4에서 lp이고 BSD에서는 lpr)에로 파이프된다.

20. here문서와 입력방향바꾸기

here문서는 mail과 sort 그리고 cat와 같은 프로그람을 위하여 즉시입력을 기억한다. 입력은 2개의 단어들이나 기호들사이에 놓인다. 첫번째 단어앞에 UNIX지령과 <<기호가 놓인다. 다음행(들)은 지령에 의해 접수되게 되는 입력으로 구성된다. 마지막행은 첫번째 단어와 꼭 같은 두번째 단어로 되여 있다. 이 단어를 최종완료자라고 하는데 입력의 끝을 표시한다. 이것의 기능은 입력을 끝내기 위해 사용되는 Control-D와 같다. 최종완료자주위에 공백이 있을수 없다. 만일 첫번째 단어앞에 <<-가 놓이면 안내타브들(타브들만이)이 최종완료자앞에 올수 있다. 보통 here문서는 대화형보다 쉘스크립트들에서 사용된다. here문서를 사용한 좋은 실례는 스크립트에서 차림표를 작성하는것이다.

형식

```
UNIX 지령 << 최종완료자
        line of input
        input
TERMINATOR
```

실례 10-64

```
(The Command Line)
1.  $ cat << FINISH# FINISH is a user-defineed terminator
2.  > Hello there $LOGNAME
3.  > The time is ${date}
    > I can't wait to see you!!!
4.  > FINISH
5.  Hello there ellie
     The time is Sun Feb 7 19:42:16 PST 2001
     I can't wait to see you!!
6.  $
```

설명

1. UNIX cat프로그람은 단어 FINISH가 행우에 단독으로 나타날 때까지 입력을 받는다.
2. here문서안에서 변수바꿔넣기가 진행된다. >는 Korn쉘의 2차재촉문이다.
3. 지령바꿔넣기가 here문서안에서 진행된다.
4. 사용자정의된 완료자인 FINISH는 cat프로그람에 대한 입력의 끝을 현시한다. 이것은 자기의 앞뒤에 임의의 공백도 가질수 없고 행우에 단독으로 나타난다.
5. cat프로그람의 출력이 현시된다.
6. 쉘재촉문이 다시 나타난다.

실례 10-65

(From the .profile File)

```
1.  primt "Select a terminal type"
2.  cat << EOF
        [1] sun
        [2] ansi
        [3] wyse50
3.  EOF
4.  read TERM
    ...
```

설명

1. 사용자가 말단형태를 선택할것을 요구한다.
2. 차림표가 현시장치에 나타난다. 이것은 here문서인데 그 의미는 여기서부터 3행에 있는 EOF가 나타날 때까지 입력이 cat지령에 주어 진다는것이다. 몇개의 echo지령들을 사용하여 같은 결과를 얻을수 있지만 보기에도 here 문서가 더 훌륭하다.
3. EOF는 사용자정의최종완료자로서 here문서의 끝을 표시한다. 그것은 주위에 공백이 없이 행의 왼쪽끝에 있어야 한다.
4. 사용자입력이 건반으로부터 입력되여 TERM에 대입된다.

실례 10-66

```
(The Command Line)
1.  $ cat <<-DONE
  >Hello there
  >What's up?
  >Bye now The time is $(date).

2.  >       DONE

3.  Hello there
    What's up?
    Bye now The time is Sun Feb 7 19:48:23 PST 2001
```

설명

1. cat프로그람은 DONE이 행우에 독립적으로 나타날 때까지 입력을 받는다. <<-연산자는 최종완료자앞에 한개이상의 타브들이 놓일수 있게 한다 (>는 쉘의 2차재촉문이다.).
2. 마지막 DONE최종완료자앞에 한개 타브가 놓인다. 행 1에 있는 첫번째 DONE으로부터 행우에 있는 마지막 DONE 사이에 있는 본문이 입력으로 cat지령에 보내 진다.
3. cat프로그람의 출력이 현시장치에 현시된다

21. 지령시간표시

time지령 time지령은 ksh의 내부지령이다. time지령은 표준오유에로 지령을 실행하는데 사용된 시간과 사용자시간 그리고 체계시간을 인쇄한다.

실례 10-67

```
1.  $ time sleep 3
     real 0m3.15s    took 3.15 seconds to run
     user 0m0.01s    sleep used its own code for .01 seconds
     sys 0m0.08s and kernel code for .08 seconds

2.  $ time ps -ef | wc -1       # time is measured for all commands in
     38                         # the pipeline
     real 0m1.03s
     user 0m0.01s
     sys 0m0.10s
```

설명

1. time지령은 지령을 실행하는데 사용된 전체 시간과 프로그람의 사용자부분이 실행되는데 걸린 시간 그리고 핵심부가 프로그람을 실행하는데 소비한 시간을 현시한다. sleep지령이 실행되는데 3.15초 걸렸다.
2. 시간이 ps지령과 wc지령에 대하여 측정된다.

22. TMOUT변수

TMOUT변수는 옹근수형이다. 이것을 설정하여 사용자들이 어떤 기간내에 지령들이 입력하도록 한다. TMOUT는 기정이 령으로 설정되여 사용자가 PS1재촉문이 현시된후에 지령들을 입력하는 시간을 제한하지 않는다. 만일 TMOUT가 령보다 큰값으로 설정되면 시간이 지난 다음에 쉘이 완료된다. 실제로 쉘을 탈퇴하기전에 60s를 더 대입해 준다.

실례 10-68

```
$ TMOUT=600
time out in 60 seconds due to inactivity
ksh: timed out waiting for input
```

설명

TMOUT변수에 600s를 설정한다. 600s동안 아무런 조작도 하지 않으면 현시장치우에 통보문이 나타난 다음 쉘을 탈퇴할 때까지 추가로 60s를 더 대입한다. 사용자가 재촉문에서 이와 같은 조작을 하면 현재쉘이 탈퇴한다.

제 2 절. Korn쉘에 의한 프로그람작성

쉘스크립트들을 작성하기 위해서는 아래에서 설명하는 몇가지 단계가 요구된다.

1. 쉘스크립트를 창조하기 위한 단계

쉘스크립트는 보통 편집기에서 작성되며 설명문이 삽입된 지령들로 구성된다.
설명문앞에는 기호 (#)가 놓인다.

첫번째 행 일반적으로 스크립트에서 행들을 실행하는 프로그람을 가리키기 위해 보통 왼쪽의 첫부분에 #!/bin/ksh를 사용한다. #!는 신비수라고 하는데 스크립트에서 행들을 해석하는 프로그람을 식별하기 위해 핵심부에 의해 사용된다. 이 행은 사용자스크립트의 제일 웃행이여야 한다. Korn쉘은 또한 쉘의 동작방식을 조종하는 호출추가선택번호를 제공한다. 이러한 추가선택을 이 장의 마지막부분에서 보여 준다.

설명문 설명문은 #기호가 앞에 놓이는 행이다. 그것들은 스크립트를 설명하는데 사용된다. 설명문을 주지 않으면 때때로 스크립트가 무슨 동작을 하는가를 리해하는것이 어렵게 된다. 설명문이 중요하지만 보통 너무 빈약하거나 지어는 전혀 쓰이지 않고 있다. 자기자신뿐아니라 다른 사람들을 위해서도 무엇을 하려고 하는가를 설명하는데 습관되여야 한다.

실행명령문과 Korn쉘구성 Korn쉘프로그람은 UNIX지령, Korn쉘지령, 프로그람구성 그리고 설명문들의 조합으로 이루어 진다.

스크립트의 이름짓기와 기억 스크립트를 이름 지을 때 의미 있는 이름과 다른 UNIX지령이나 별명들과 중복되지 않는 이름으로 짓는것이 좋다. 실례로 어떤 간단한 시험처리를 진행하려고 하기때문에 스크립트를 test로 이름 지우려고 할수 있지만 이렇게 하면test가 내부지령이므로 정확치 않은 test가 실행될수 있다.더우기 파일을 foo, goo, boobar 등으로 이름 지으면 며칠이 지나도 사용자는 스크립트안에 무엇이 있는지 알수 없다. 스크립트를 시험하여 그것이 오유가 없다는것을 확인한 다음 스크립트를 기억할수 있는 등록부를 만들고 경로를 설정하여 스크립트들이 등록부가지의 임의의 곳에서 실행될수 있게 한다.

실례 10-69

1. $ **mkdir ~/bin**
2. $ **mv myscript ~/bin**

(In *.profile*)

3. export PATH=${PATH}:~/bin
4. $ **. .profile**

설명

1. 스크립트들을 기억시키는 일반위치는 홈등록부밑에 있는 bin이라는 등록부로 된다.
2. myscript라는 스크립트를 새로운 bin등록부에로 이동한다.
3. 새로운 등록부를 .profile초기화파일안에 있는 PATH변수에 첨가한다.

4. dot지령은 .profile을 현재환경에서 실행되게 하여 새로운 설정들을 가능하게 하기 위해 등록탈퇴하였다가 다시 등록가입하지 않아도 된다.

스크립트를실행가능하게 하기 파일을 작성할 때에는 자동적으로 실행허가가 주어지지 않는다(umask가 어떻게 설정되였는가에는 관계없이). 스크립트를 실행시키기 위해서는 이 허가가 필요하다. 실행허가를 설정하기 위해서는 chmod지령을 사용한다.

실례 10-70

1. $ **chmod +x myscript**
2. $ **ls –lF myscript**
   *-rwxr--xr--x 1 ellie 0 Jul 12 13：00 joker**

설명

1. chmod지령은 사용자, 그룹 등에 실행허가를 설정하기 위해 사용된다.
2. ls지령의 출력은 모든 사용자가 joker파일에 대한 실행허가를 가진다는것을 가리킨다. 파일의 끝에 있는 별표(*)도 이것이 실행가능한 프로그람이라는 것을 가리킨다.

ksh의 인수로 스크립트를 사용하기 스크립트가 실행가능하게 하지 않으면 ksh지령에 스크립트를 인수로 넘기여 실행할수 있다.

실례 10-71

(The Command Line)
$ **ksh myscript**

설명

ksh프로그람에 인수로 스크립트이름을 주면 스크립트가 실행되며 #!행은 필요없거나 지어는 사용되지 않는다.

스크립트작성대화조종 실례 10-72에서 사용자는 편집기에서 스크립트를 작성한다. 파일을 기억한 다음 실행허가가 설정되고 스크립트가 실행된다. 프로그람에 오유가 있으면 Korn쉘은 즉시 응답한다.

실례 10-72

(The Script)
1. #!/bin/ksh
2. *# This is the first Korn shell program of the day.*
   # Scriptname: greetings
   # Written by: Karen Korny
3. print "Hello $LOGNAME, it's nice talking to you."
4. print "Your present working directory is $(pwd)."
   print "You are working on a machine called $ (uname –n)."

```
        print "Here is a list of your files."
    5.  ls        # List files is the present working directory
        print "Bye for now $LOGNAME. The time is $(date +%T)!"
```

(The Command Line)
$ **chmod +x greetings**
$ **greetings**
3. *Hello karen, it's nice talking to you.*
4. *Your present working directory is /home/lion/karen/junk*
   Your are working on a machine called lion.
   Here is a list of your files.
5. *Afile cplus letter prac*
   Answerbook csrog library pracl
   bourne joke notes perl5
   *Bye for now karen. The time is 18:05:*07

설명

1. 스크립트의 첫 행 #!/bin/ksh는 이 프로그람에서 행을 실행하는것이 무슨 해석기인가를 핵심부에 알려준다.
2. 설명문들은 #로 시작되는 실행되지 않는 행들이다. 그것들은 행자체에 놓일 수도 있고 지령 다음에 삽입해 넣을수도 있다.
3. print지령은 쉘에 의해 변수바꿔넣기를 진행한 다음 현시장치에 행을 현시한다.
4. print지령은 쉘에 의해 지령바꿔넣기를 한 다음 현시장치에 행을 현시한다.
5. ls지령이 실행된다. 행우에서 기호(#)다음의 설명문은 쉘에 의해 무시된다.

2. 사용자입력읽기

read지령은 행바꾸기가 있을 때까지 말단이나 파일로부터 입력을 얻기 위해 사용된다. Korn쉘은 read지령에 대한 몇가지 첨가추가선택들을 제공한다. 여러가지read 형식에 대하여 표 10-20에 보여 주었다. read추가선택들에 대해서는 표 10-21에서 보여 주었다.

표 10-20. read형식

형식	의 미
Read answer	표준입력으로부터 행을 읽고 그것을 변수 answer에 대입한다.
Read first last	표준입력으로부터 첫 공백이나 행바꾸기가 나타날 때까지 행을 읽고변수 first에 첫 last번째 단어를 대입하고 변수 행의 나머지부분을 넣는다.
read response? “Do you feel okey?”	표준오유에 문자렬 “Do you feel okey?”를 현시하고 응답이 있을 때까지 대기한다. 그다음 응답이 있으면 변수 response에 대입한다. read지령의 형식은 오직 한개 변수만을 요구하고 접수한다. 사용자가 입력하면 행바꾸기가 있을 때까지 응답을 기억한다.

(표계속)

형식	의 미
Read –u3 line	변수 line에 파일서술자3으로부터 행을 읽어 넣는다.
Read	내부변수 REPLY에 입력을 읽어 넣는다.

표 10-21. read추가선택

추가선택	의 미
-r	문자그대로 \n을 행바꾸기문자로 간주한다.
-s	리력파일안에 행을 복사한다.
-un	파일서술자 n으로부터 읽는다. 기정은 fd0 아니면 표준입력이다.
-p	협동프로쎄스로부터 입력행을 읽는다.
	Ksh의 판본이 1988이상이 아닐 때
-a	첨수 0으로 시작하는 배렬안에 항목들을 기억한다.
-t sec	사용자응답시간의 한계(단위:s)를 준다.
-d char	입력끝경계를 지적한다. 기정은 행바꾸기이다.

실례 10-73

(The Script)

```
   #!/bin/ksh
   # scriptname: nosy
   print –n "Are you happy?  "
1. read answer
    print "$answer is the right response. "
    print –n "What is your full name?  "
2. read first middle last
   print "Hello $first"
          print –n "Where do you work?  "
3. read
4. print I guess $REPLY keeps you busy!
5. read place?"Where do you live?  "
   # New ksh read and print combined
   print Welcome to $place, $first $last
```

(The output)

```
$ nosy
Are you happy?  Yes
```

1. *Yes is the right response.*
2. *What is your full name?* ***Jon Jake Jones***

Hello Jon

3. *Where do you work? Tandem*
4. *I guess Tandem keeps you bush!*
5. *Where do you live?* ***Timbuktu***
   Welcome to Timbuktu, Jon Jones

설명

1. read지령은 사용자입력행을 접수하고 변수 answer에 입력을 대입한다.
2. read지령은 사용자입력행을 접수하고 변수 first에는 첫번째 입력단어를 변수 middle에 두번째 입력단어를 대입하며 변수 last에 행의 끝까지 나머지 단어들을 대입한다.
3. 인수가 없는 read지령은 사용자입력행을 접수하고 내부변수 REPLY에 입력을 대입한다.
4. 쉘이 변수바꿔넣기를 한 다음 print함수는 내부변수 REPLY의 값을 문자렬로 출력한다.
5. read지령 다음의 변수에 물음표(?)가 붙어 있으면 물음표(?)다음 물음표들은 재촉문으로 현시된다. 사용자입력은 변수 place에 기억된다.

읽기와 파일서술자 체계가 초기넣기될 때 흐름이라는 3개의 파일 표준입력, 표준출력 그리고 표준오유)들이 열어 지며 파일서술자배렬에로 대입된다. 우선 3개의 파일서술자들인 0, 1, 2는 각각 표준입력, 표준출력 그리고 표준오유에 대응된다. 다음에 사용할 수 있는 파일서술자는 파일서술자 3이다. u추가선택은 read지령이 파일서술자로부터 직접 읽도록 허용한다.

실례 10-74

(The Command Line)

1. $ **cat filex**
   Captain Kidd
   Scarlett O'Hara

2. $ **exec 3< filex**
   # filex is assigned to file descriptor 3 for reading

3. $ read –u3 name1
   # read from filex and store input in variable, name1

4. $ print $name1
   Captain Kidd

5. **$ read –u3 name2**
   $ print $name2
   Scarlett O'Hara

6. $ **exec 3<&-** *# Close file descriptor 3*

7. $ **read –u3 line**
   ksh: read: bad file unit number

설명

1. filex의 내용들이 현시된다.
2. exec지령은 filex로부터 읽기 위해 파일서술자3을 여는데 사용된다.
3. read지령은 장치 3(파일서술자3, filex)으로부터 직접 한 행을 읽고 변수 name1에 행을 대입한다.
4. name1에 기억된 행이 현시된다.
5. 파일 filex가 여전히 열려져 있고 read지령은 파일로부터 다음행을 읽으며 변수 name2에 행을 기억한다.
6. 파일서술자3(장치3)이 닫겨 진다. filex를 당분간 열지 않는다.
7. 파일서술자3(filex)이 닫겨 져 있기때문에 변수 line에 파일서술자로부터 입력을 읽으려고 시도하면 read지령은 실패한다.

파일로부터 읽기 실례 10-75에서는 while순환과 함께 read지령을 사용한다. 순환은 한번에 한 행씩 파일을 모두 조사한다. 파일의 끝에 도달하면 순환은 끝난다. 파일은 읽기 위해 파일서술자(장치들)들과 함께 열어 진다.

실례 10-75

(The Files)
$ **cat names**
Merry Melody

Nancy Drew
Rex Allen
$ **cat addresses**
150 piano Place
5 Mystery Lane
130 Cowboy Terrace
- -

(The Script)
```
  #!/bin/ksh
# Scriptname: readit

2.  while  read  –u3  line1  &&  read  –u4  line2
     do
3.         print "$line1:$line2"
4.  done  3<$1  4<$2
```
- -

(The Command Line)
5. $ ***readit names addresses***

Merry Melody:150 Piano Place
Nancy Drew:5 Mystery Lane
Rex Allen:130 Cowboy Terrace
- -

설명

1. 2개의 파일 즉 names와 addresses의 내용이 현시된다.
2. while순환이 시작된다. read지령은 파일서술자3(장치 3)으로부터 입력행을 읽는데 성공하면 파일서술자 4로부터 다른 행을 읽는다. 4행에서 파일서술자들에 파일이름이 대입된다. 파일이름들은 인수들이나 혹은 위치파라메터 1과 2로 넘겨 진다.
3. 첫번째 변수의 값, 두점(:), 두번째 변수의 값이 현시된다.
4. 파일서술자 3에 대입된 입력은 첫번째 지령행인수 즉 name이다. 파일서술자 4에 대입된 입력은 두번째 지령행인수 즉 addresses이다.
5. 스크립트는 지령행인수(2개의 파일이름들)들과 같이 실행된다.

3. 산수연산

Korn쉘은 옹근수연산과 류동소수점연산이라는 두가지 산수연산을 제공하지만 류동소수점산수연산은 오직 Korn쉘판본이 1988이상이 아닐 때에만 쓸수 있다. typeset지령은 입력을 대입하기 위해 사용된다. typeset지령에 대해서는 표 10-22에서 보여 주었다.

표 10-22. typeset지령과 산수연산

typeset지령	별 명	의 미
typeset –i 변수	Integer변수	변수는 오직 옹근수이여야 한다.
typeset –i#		#는 옹근수를 위한 기초수이다.
ksh판본이 1988이상이 아닐 때		
typeset –F변수		류동소수점수대입
typeset –E변수	float변수	류동소수점수대입

옹근수형 변수는 typeset –i지령이나 별명 즉 integer를 사용하여 옹근수로 선언할 수 있다. 사용자가 어떤 문자렬값을 대입하면 ksh는 오유를 되돌려보낸다.

사용자가 류동소수점수를 대입하면 옹근수부와 소수부가 되돌려진다. integer별명은 typeset –i대신에 사용할수 있다. 수값은 2진수, 8진수 그리고 16진수와 같은 여러가지 기초수들로 표현될수 있다.

실례 10-76

1. $ **typeset –i num** or **integer num** *# integer is an alias for*
   # typeset –i
2. $ **num=hello**
   /bin/ksh: hello:bad number
3. $ **num=5 + 5**

/bin/ksh: +:not found

4. $ **num=5 + 5**
   $ echo $num
   10

5. $ **num=4*6**
   $ echo $num
   24

6. $ **num="4 * 6"**
   $ echo $num
   24

7. $ **num=6.789**
   $ echo $num
   6

설명

1. typeset지령에 –i추가선택을 주면 옹근수변수 num을 작성한다
2. 옹근수변수 num에 문자렬 hello를 대입하면 오유가 발생한다
3. 옹근수변수 num에 문자렬 hello를 대입하면 오유가 발생한다.
4. 공백(( ))연산자들을 사용하지 않는한 인용팔호안에 넣거나 소거해야 한다.
5. 공백이 소거되고 산수연산이 실행된다.
6. 곱하기연산이 진행되고 결과를 num에 대입한다.
7. 공백이 인용부호안에 있기때문에 곱하기연산이 실행되며 쉘이 통용기호(*) 확장을 하지 않게 한다. 변수가 옹근수로 설정되여 있기때문에 소수부를 떼 버린다.

여러가지 기초수의 사용 typeset지령에 –i추가선택과 기초수를 같이 주면 10진수(기초수 10), 8진수(기초수 8) 그 밖의 수들을 현시할수 있다.[6]

실례 10-77

1. $ **num=15**
2. $ **typeset –i2 num** *# binary*
   $ **print $num**
   2#1111

3. $ **typeset –i8 num** *# octal*
   $ print $num
   8#17
4. $ typeset –i16 num # hex
   $ **print $num**
   16#f

5. $ **read number**

6. 36 보다 큰 기초수들은 1988 이후의 korn 쉘에서 쓸수 있다.

```
        2#1101
        $ print $number
        2#1101
    6.  $ typeset –i   number
        $ print $number
        2#1101
    7.  $ typeset –i10   number      # decimal
        $ print $number
        13
    8.  $ typeset –i8   number       # octal
        $ print $number
        8#15
```

설명

1. 변수 num에 값 15를 대입한다.
2. typeset지령은 2진수형태로 값을 변환한다. 기초수(2)다음에 기호(#) 그리고 2진수값형태로 현시된다.
3. typeset지령은 8진수로 값을 변환하고 8진수값을 현시한다.
4. typeset지령은 16진수로 값을 변환하고 16진수값을 현시한다.
5. read지령은 사용자로부터 입력을 접수한다. 입력은 2진수형태로 넣어 지며 변수 number에 기억되고 2진수형태로 현시된다.
6. typeset지령은 옹근수형으로 수를 변환한다. 여전히 2진수형태로 현시된다.
7. typeset지령은 10진옹근수형으로 수를 변환하고 그 값을 현시한다.
8. typeset지령은 8진수형으로 수를 변환하고 그 값을 현시한다.

옹근수의 목록화 typeset지령에 오직 –i인수만을 주면 옹근수들과 그 값을 다음과 같이 목록화한다.

```
$ typeset –i
ERRNO=2
LINENO=1
MAILCHECK=600
OPTIND=1
PPID=4881
RANDOM=25022
SECONDS=47366
TMOUT=0
N=5
Number=#15
```

산수연산자와 let지령 let지령은 옹근수산수연산을 진행하는 Korn쉘 내부지령이다. 이것은 Bourne쉘 옹근수검사를 치환한다. let지령을 사용하는데서 좋은 방법은 (( ))연산자를 함께 사용하는것이다.

표 10-23. let연산자

연산자	의 미
-	부수
!	론리부정
~	bit반전
*/%	곱하기, 나누기, 나머지
+ -	더하기, 덜기
<< >>	bit왼쪽 밀기, bit오른쪽 밀기
<= >= <> == !=	비교연산자
& ^|	bit적, 안맞음론리합
&& || !	론리적, 론리합, 부정
=	대입
*= /= %= += -= <<= >>= &= ^= !=	지름대입

주의: ++와 - - 연산자들은 1988이후의 ksh판본에서만 지원된다.

실례 10-78

1. $ **i=5**
2. $ **let i=i+1**
   $ **print $i**
   6
3. $ **let "I = i + 2"**
   $ **print I**
   8
4. $ **let "i+=1"**
   $ **print $i**
   9

설명

1. 변수 i에 값 5를 대입한다.
2. let지령은 i값에 1을 더한다. 산수연산이 진행될 때 변수바꿔넣기를 위해 $가 요구되지 않는다.
3. 인수들에 공백이 포함되여 있으면 인용부호가 필요된다.
4. 지름연산자 즉 +=는 i의 값에 1을 더하기 위해 사용된다.

실례 10-79

```
(The Command Line)
1.  $ ((  i  =  9  ))
2.  $ ((  i  =  i  *  6 ))
     $ print  $i
     54
3.  $ ((  i  >  0  &&  i  <=  10  ))
4.  $ print  $?
     1
     $ j = 100
5.  ((  i  <  j  ||  i  ==  5  ))
6.  $ print  $?
     0
7.  $ If ((  i  < j  &&  i  ==  54 ))
     >then
     >print  True
     >fi
    True
    $
```

설명

1. 변수 i 에 값 9를 대입한다. (( ))연산자들은 let지령의 한가지 형식이다. 표현식이 2중괄호안에 들어 있기때문에 연산자들속에서 공백을 사용할수 있다. 변수 i 에 값 9를 대입한다. (( ))연산자들은 let지령의 한가지 형식 이다. 표현식이 2중괄호안에 들어 있기때문에 연산자들속에서 공백을 사용할수 있다.
2. 변수 i에 i*6의 결과를 대입한다.
3. 수값표현식을 시험한다. 2개의 수값표현식이 모두 참이면 탈퇴상태 0을 되돌려 준다.
4. 특수한 물음표(?)변수는 마지막실행지령(let지령)의 탈퇴상태를 가진다. 값이 령이 아니기때문에 지령이 실패한다는것을 알수 있다(거짓으로 평가된다.).
5. 수값표현식을 평가한다. 표현식가운데서 하나만 참이면 탈퇴상태 0을 되돌려 준다.
6. 특수한 물음표(?)변수는 마지막실행지령(let지령)의 탈퇴상태를 가진다. 탈퇴상태값이 0이기때문에 지령이 성공한다는것을 알수 있다(참으로 평가된다.).
7. if조건지령이 let지령의 앞에 있다. 지령이 완성되기를 기다리는 동안 2차재촉문이 현시된다. 탈퇴상태가 령이면 then 명령문다음에 있는 지령들이 실행된다. 그외에는 1차재촉문으로 되돌아 온다

4. 위치파라메터와 지령행인수

$지령행인수들은 위치파라메터와 함께 스크립트안에서 참조될수 있다. 례를 들어 $1은 첫번째 인수, $2은 두번째 인수 그리고 $3은 세번째 인수에로 각각 설정된다. 위치파라메터들은 set지령에 의해 재설정될수 있다. 위치파라메터들을 표 10－24에서 보여 준다.

표 10－24. 위치파라메터

변 수	기 능
$0	스크립트의 이름을 참고한다.
$#	위치파라메터들의 개수값을 가진다.
$*	모든 위치파라메터들의 목록을 포함한다.
$@	$*와 같다. 2중인용부호들이 있을 때 제외된다.
"$*"	한개의 인수로 확장한다. "$1 $2 $3"과 같다.
"$@"	서로 분리된 인수로 확장한다. "$1" "$2" "$3"과 같다.

set지령과 위치파라메터 set지령은 위치파라메터들을 설정한다. 위치파라메터들이 이미 설정되여 있으면 set지령은 낡은 목록에서 모든 값들을 소거하면서 그것들을 재설정한다. 모든 위치파라메터들을 해제하기 위해서는 set - - 지령을 사용해야 한다.

실례 10-80

```
(The Script)
    $ cat args
    #!/bin/ksh
    # Script to test command line arguments
1.  print The name of this script  i s $0.
2.  print The arguments are $*.
3.  print The first argument  i s $1.
4.  print The second argument  i s $2.
5.  print The number of arguments  i s $#.
6.  o ldparameters=$*
7.  s et Jake Nicky Scott
8.  Print All the positional parameters are $*.
9.  Print The number of positional parameters  i s $#.
10. print $oldparameters
11. s et - -
12. print Good-bye for now, $1.
13. s et $oldparameters
14. print $*
```

(The Output)

$ **args a b c d**

1. *The name of this script i s* **args.**
2. *The arguments are* **a b c d.**
3. *The first argument i s* **a**
4. *The second argument i s* **b.**
5. *The number arguments i s* **4.**
8. *All the positional parameters are **Jake Nikcy Scott.***
9. *The number of positional parameters i s* **3.**
10. *a b c d*
12. *Good-bye for now , .*
14. *a b c d*

$

설명

1. 스크립트의 이름이 $0변수에 기억된다.
2. $*(그리고 $@)는 모든 위치파라메터들을 표현한다.
3. $1은 첫번째 위치파라메터를 표현한다(지령행인수).
4. $2은 두번째 위치파라메터를 표현한다.
5. $#는 위치파라메터들의 총개수이다(지령행인수).
6. 변수 oldparameters에 모든 위치파라메터들($*)을 대입한다. 다음 원래의 파라메터들을 다시 얻으려면 set $oldparameters를 입력해야 한다.
7. set지령은 위치파라메터들을 재설정한다. set지령이 정확히 원래의 모든 set 파라메터들을 소거한다. Jake는 $1에 대입되고 Nicky는 $2에 대입되며 그리고 Scott는 $3에 대입된다.
8. 새로운 위치파라메터들이 출력된다.
9. 위치파라메터들의 개수가 출력된다.
10. 원래의 파라메터들이 변수 oldparameters에 기억된다. 그것들이 현시된다.
11. 모든 파라메터들이 대입되지 않는다.
12. $1은 값을 가지지 않는다. 파라메터목록이 set - -지령에 의해 소거된다.
13. set지령에 의해 oldparameters변수안에 있는 값들을 바꿔넣기함으로써 새로운 파라메터를 대입한다.
14. 모든 위치파라메터들이 출력된다.

실례 10-81

(How $* and $@ Differ)

1. $ **set ‘apple pie’ pears peaches**
2. $ **for i i n $***

>do

```
>echo $ i
>done
apple
pie
pears
peaches
```

3. $ set ‘apple pie’ pears peaches

4. $ for i i n “$*”
   > do
   > echo $ i
   > done
   apple pie pears peaches

5. $ set ‘apple pie’ pears peaches
6. $ for i i n $@
   > do
   > echo $ i
   > done

   apple pie
   pears
   peaches
7. $ set ‘ apple pie ‘ preas peaches
8. $ for i i n “$@”   # *At last!!*
   > do
   > echo $ i
   > done
   apple pie
   pears
   peaches

설명

1. 위치파라메터들이 설정된다. $*가 확장될 때 인용부호는 벗겨 지며 apple pie는 2개의 분리된 단어로 된다. for순환은 변수 i 에 매 단어들을 대입한 다음 i 값을 출력한다. 순환의 매 단계마다 왼쪽의 단어가 밀기되며 다음 단어가 i 에 대입된다.
2. $*가 2중인용부호안에 있으면 왼쪽에 있는 모든 단어들은 개개의 단일문자렬로 된다. 모든 문자렬이 변수 i에 대입된다.
3. 위치파라메터들을 설정한다.
4. 2중인용부호안에 $*가 들어있기때문에 입력파라메터목록은 개개의 물음표로 된다.

5. 위치파라메터들을 설정한다.
6. 인용괄호가 없기때문에 $@는 $*와 같은 특성을 가진다.
7. 위치파라메터를 설정한다.
8. 2중인용괄호안에 $@가 있기때문에 매 위치파라메터들은 인용괄호안에 있는 문자렬로 취급된다. 목록은 “apple pie”, ”pears” 그리고 “peaches”로 이루어 진다. 순환이 모두 반복되기때문에 인용괄호안의 매 단어들이 i에 대입된다.

5. 탈퇴상태의 시험과 $?변수

?변수는 탈퇴된 마지막지령의 탈퇴상태를 현시하는 수값(0으로부터 255까지)을 가진다. 탈퇴상태가 0이면 지령이 성공적으로 탈퇴한다는것이고 0이 아니면 지령이 여러가지 원인으로 실패한다는것을 가리킨다.

사용자는 지령의 탈퇴상태를 시험할수 있으며 표현식의 탈퇴상태를 시험하기 위해 test지령을 사용한다. 다음의 실례에서는 어떻게 탈퇴상태를 시험하는가를 보여 준다. 단일중괄호는 Bourne쉘에서 사용되며 비록 Korn쉘에서 완전히 접수되여도 Dr.Korn은 표현식시험에 새로운 2중중괄호([[ ]])를 제공한다.

실례 10-82

```
(The Command Line)
1.  $ name=Tom
2.  $ grep “$name” datafile
    Tom savage:408-124-2345
3.  $ print   $?
    0       # Success!

4.  $ test $name = Tom

5.  $ print $
    0                               # Success
6.  $ test $name != Tom
$ print $?
    1                               # Failure

7.  $ [ $name =   Tom   ]         # Brackets Instead of The test command
8.  $ print $?
    0

9.  $ [[ $name = [Tt] ?m ]]     # new ksh test command
  10.  $ print $?
    0
```

설명

1. 문자렬 Tom을 변수 name에 대입한다.

2. grep지령은 datafile안에서 문자렬 Tom을 검색한다. 검색이 성공하면 검색된 행을 현시한다.
3. $?에 의해 접수된 물음표(?)변수는 마지막지령실행의 탈퇴상태값을 가리킨다. 이 경우는 grep지령의 탈퇴상태에 해당된다. grep지령은 성공한다.
4. test지령은 문자렬들과 수값들을 평가하기 위해 사용된다. 또한 파일시험도 한다.
5. 표현식이 참이면 탈퇴상태 0을 되돌려 주고 표현식이 거짓이면 0이 아닌 탈퇴상태를 되돌려 준다. 같기부호(=) 량옆에는 공백을 주어야 한다.
6. name의 값이 Tom인가를 시험한다. $name이 Tom으로 평가되기때문에 test지령은 탈퇴상태0을 되돌려 준다.
7. name의 값이 Tom인가를 시험한다. $name이 Tom이 아니기때문에 test지령은 탈퇴상태1을 되돌려 준다.
8. 중괄호[ ]부호를 test지령대신에 사용한다. 첫번째 중괄호다음에는 공백을 주어야 한다. $name이 문자렬 Tom인가를 알기 위해 표현식을 평가한다. 시험의 탈퇴상태가 0이다. 시험은 $name이 문자렬 Tom과 같기때문에 성공한다.
9. 새로운 Korn쉘의 test지령 즉 2중중괄호[[를 사용한다. 새로운 test지령은 쉘파라메터확장을 허용한다. 변수가 Tom, tom, Tim등이면 시험은 성공적인 탈퇴상태로 0을 되돌려 준다.
10. 변수 name에 문자렬 T 혹은 t로 시작하여 m으로 끝나는 문자렬이 대응되면 성공적인 탈퇴상태($?)를 나타내는 0을 되돌려 준다.

6. 조건구성체와 흐름조종

조건지령들은 조건이 성공인가 아니면 실패인가에 따라 몇개의 파제들을 실행할수 있다. If지령은 간단한 판정만들기 형식이다. If/else지령들은 두 방향의 판정구성을 허락하며 if/elif/else지령들은 여러 방향의 판정구성을 허락한다. Korn쉘은 if다음의 지령을 수행한다. 지령은 체계지령 혹은 내부지령이여야 한다. 지령의 탈퇴상태는 조건을 평가하기 위해 사용된다. 표현식을 평가하기 위해 내부지령 test를 사용한다. 이 지령도 역시 중괄호와 2중중괄호로 련결된다. Bourne쉘은 단일중괄호의 쌍인 [와]기호안에 표현식을 넣을것을 요구한다. Korn쉘은 표현식을 평가하는데 더 고급한 방법을 가지고 있다. 표현식은 2중중괄호안에 넣어야 한다. 단일중괄호안에서 통용기호의 확장은 허용되지 않는다. 2중중괄호(오직 Korn쉘에서만)와 함께 통용기호확장가 사용되며 새로운 연산자들이 첨부되였다. 지령의 결과는 성공을 가리킬 때는 탈퇴상태 0으로 검사되고 실패를 가리킬 때는 령이 아닌 탈퇴상태가 검사된다.

낡은 test지령 test지령은 조건식이 참인가 거짓인가를 평가하기 위해 사용된다. 참일 때는 탈퇴상태 0을 되돌려 주고 거짓일 때는 령이 아닌 탈퇴상태를 되돌려 준다. test지령 아니면 중괄호를 사용할수 있다. Korn쉘은 2중중괄호로 표현식을 검사하는 새로운 방법을 소개한다. Bourne쉘의 뒤방향일치성을 위해 test의 낡은 형태인 test지령 아니면 단일중괄호를 사용할수 있다. 그러나 Korn쉘프로그람작성자들을 위한 개선된 방법에는 2중중괄호라는 새로운 test가 있다. 검사연산자들에 대한 완성된 목록(낡은것과 새로운것 둘다)을 표 10-25에 보여 준다.

표 10-25. 검사와 론리연산자

검사	무엇을 검사하는가
문자렬검사:	
문자렬 1=문자렬 2	문자렬 1이 문자렬 2와 같은가?
문자렬 1!=문자렬 2	문자렬 1이 문자렬 2와 같지 않은가?
문자렬	문자렬이 null값이 아닌가?
-z문자렬	문자렬의 길이가 0인가?
-n문자렬	문자렬의 길이가 0이 아닌가?
	실례:
	test –n $word 혹은 [-n $word;]
	test tom = sue 혹은 [tom = sue]
옹근수검사(낡은형 시험은 Bourne쉘에서 사용된다.)	
Int1 –eq int2	Int1이 int2와 같은가?
Int1 –ne int2	Int1이 int2와 같지 않은가?
Int1 –gt int2	Int1이 int2보다 큰가?
Int1 –ge int2	Int1이 int2와 같거나 큰가?
Int1 –lt int2	Int1이 int2보다. 작은가?
Int1 –le int2	Int1이 int2와 같거나 작은가?
론리연산자들(낡은 형검사)	
!	부정연산자
-a	적연산자
-o	합연산자
파일시험(낡은 형검사)	
-b파일이름	블로크형파일
-c파일이름	문자형파일
-d파일이름	등록부가 존재
-f파일이름	파일이 존재하며 등록부가 아니다.
-g파일이름	그룹ID가 설정된다.
-h파일이름	기호련결
-k파일이름	련결bit가 설정된다.
-p파일이름	파일이 파이프로 이름 지어 진다.
-r파일이름	파일이 읽기전용이다.
-s파일이름	파일의 크기가 령이 아니다.

(표계속)

검사	무엇을 검사하는가
-u파일이름	사용자IDbit를 설정한다.
-w파일이름	파일이 쓰기전용이다.
-x파일이름	파일이 실행가능하다.

새로운 test지령 2중중괄호([[...]])인 복합 test지령과 함께 추가연산자들을 사용할수 있다. 통용기호를 문자렬검사에 사용하며 낡은 test지령의 여러가지 오유들이 지우기된다. 새로운 문자렬검사연산자들을 표 10-26에 보여 준다.

표 10-26. 문자렬검사(새로운 형의 검사)

문자렬검사연산자	검 사
문자렬 = 패턴	문자렬이 패턴에 대응된다.[a]
문자렬 != 페턴	문자렬이 패턴에 대응되지 않는다.
문자렬 1 < 문자렬 2	문자렬 1의 ASCII값이 문자렬 2보다 작다.
문자렬 1 > 문자렬 2	문자렬 1의 ASCII값이 문자렬 2보다 크다.
-z문자렬	문자렬의 길이가 0이다. 즉 null 파라메터이다.
-n문자렬	문자렬의 길이가 령이 아니다. 즉 null 파라메터가 아니다.

실례 10-83

```
(The Script)
read answer
1. if  [[  $answer  =  [Yy]*  ]]        # Test for Yes or yes or Y or y,etc.
     then...
     Example:

(The Script)
quess=Noone
2. if  [[  $quess  !=  [Nn]o@(one|body)  ]]       # Test for Noone,noone
    then...                                       # or Nobody, nobody...

    Example:
(The Command Line)
3.  [[  apples  <  oranges  ]]
     print $?
    0
4.  [[  apples  >  oranges  ]]
     print  $?
     1
5.  $  name  =  "Joe Shmoe"
```

[a] 1988 이후의 판본에서는 ==연산자를 허용한다.

```
   $ [ $name = "Abe Lincoln" ]          # old style
   ksh: Shmoe: unknown test operator
6. $ [[ $name = "Abe Lincoln" ]]        # new style
   $ echo $?
   1
```

설명

1. 사용자로부터 읽어 넣은 변수 answer가 Y 또는 y로 시작하는 문자렬인가를 검사한다.
2. 변수 guess를 검사한다. 변수가 N 또는 n으로 시작하며 다음에 o가 있고 다음에 정확히 one 혹은 body로 이루어 진 문자렬 실례로 none 또는 nobody가 아니면 지령이 실행된다.
3. 문자렬 apples가 ASCII 순서에서 문자렬 orange전에 있으면 검사된다. 결국 검사한다.
4. 문자렬 apples가 ASCII 순서에서 문자렬 orange다음에 있으면 검사된다. 결국 검사하지 않는다.
5. 낡은 형의 test지령으로 변수 name을 개개의 단어들로 분리한다. 같기(=)연산자는 왼쪽 연산수로 한개 문자렬을 요구하기때문에 test지령이 실패한다. 이 문제를 해결하기 위해서는 변수를 2중인용부호안에 넣어야 한다.
6. 새로운 형의 test지령에서는 변수를 개개의 단어들로 분리하지 않는다. 그러므로 2중인용부호가 $name에 필요 없다.

2진연산자에 의한 파일시험 파일을 검사하기 위한 2진연산자들은 2개의 연산수를 요구한다(즉 연산자의 량쪽에 있는 파일). 2진파일연산자들의 목록을 표 10—27에 보여준다.

표 10-27. 2진파일시험와 론리연산자

연산자	검 사
2진파일시험	
파일 1 –nt파일 2	파일 1이 파일 2보다 새것이면 참이다.
파일 1 –ot파일 2	파일 1이 파일 2보다 낡은것이면 참이다.
파일 1 –ef파일 2	파일 1이 파일 2와 다른 이름이면 참이다.

론리연산자 C와 비슷한 Korn쉘은 표현식이 정확한가 또는 거짓인가를 판정하는 론리검사를 제공한다. 그것들을 표 10-28에 보여 준다.

표 10−28. 론리연산자

연산자	검 사
&&	론리적연산자는 &&의 왼쪽 표현식을 평가한다. 참이면 &&의 오른쪽 표현식을 평가하는데 역시 참이여야 한다. 한개의 표현식이라도 거짓이면 표현식은 거짓으로 된다. &&연산은 –a와 치환된다. 실례로 ((($x && $y)>5))이다.

(표계속)

연산자	검 사
||	론리합연산자는 ||연산자의 왼쪽에 있는 표현식을 평가한다. 참이면 표현식은 참으로 된다. 거짓이면 ||연산자의 오른쪽에 있는 표현식을 평가하며 참이면 표현식은 참으로 된다. 오직 두 표현식이 모두 거짓일 때 표현식은 거짓으로 평가된다. ||연산자는 –o와 치환된다. 실례로 (((&x ||$y) >5))이다.

파일시험 Korn쉘은 존재, 형, 허가 등과 같은 파일속성들을 검사하기 위한 test내부지령들을 제공한다. 파일시험추가선택들을 표 10-29에 보여 준다.

표 10-29. 파일시험(새로운 test기발)

검사기발	검 사
오직 Korn쉘에서만	
-a파일	파일이 존재한다.
-e파일	파일이 존재한다(1988이상이 아닌 판본들에서).
-L파일	파일이 존재하며 기호련결된다.
-O파일	파일자체이다.
-G파일	그룹ID가 파일에 설정된것과 같다.
-S파일	파일이 존재하며 련결된다.
Bourne과 Korn쉘에서	
-r파일	파일이 존재하며 읽기가능하다.
-w파일	파일이 존재하며 쓰기가능하다.
-x파일	파일이 존재하며 실행가능하다.
-f파일	파일이 존재하며 등록부가 아니다.
-d파일	파일이 존재하며 등록부이다.
-b파일	파일이 존재하며 블로크형특수파일이다.
-c파일	파일이 존재하며 문자형특수파일이다.
-p파일	파일이 존재하며 파이프로 이름 지어 진다.
-u파일	파일이 존재하며 사용자ID를 설정한다.
-g파일	파일이 존재하며 그룹 ID를 설정한다.
-k파일	파일이 존재하며 련결bit가 설정된다.
-s파일	파일의 크기가 령이 아니다.

실례 10-84

```
(The Script)
1.  file=/etc/passwd
```

```
2.  if  [[  -f    $file && (-r $file \ \    -w $file)   ]]
    then
3.     print     $file is a plain file and is either readable or   writeable
      fi
```

설명

1. 변수 file에 /etc/passwd를 대입한다.
2. 파일시험연산자들은 파일이 정확히 파일이면서 읽기가능하거나 쓰기가능하면 검사된다. 괄호는 그룹별로 묶기 위해 사용된다. 낡은 형의 test지령에서 소괄호 ()는 \ 기호를 벗어 나게 한다.
3. 만약 파일이 정확히 파일이면서 읽기가능하거나 쓰기가능하면 행이 실행된다.

If지령 간단한 조건형태는 if지령이다. If예약어 다음의 지령들이 실행되며 탈퇴상태를 되돌려 준다. 탈퇴상태가 령이면 지령이 성공이라는것을 의미하며 then예약어 다음의 명령문들이 실행된다.

C쉘과 C언어에서 if지령 다음의 표현식은 Boolean형의 표현식이다. Bourne와 Korn쉘들에서 if다음의 명령문들은 지령 혹은 지령들의 그룹이다. if행에서 마지막지령의 탈퇴상태는 계속하겠는가 아니면 안하겠는가를 확정하는데 사용하는데 then명령문다음의 지령들을 실행한다. fi는 then다음의 실행하려고 하는 지령목록을 끝낸다. 탈퇴상태가 령이 아니면 여러가지 원인으로 지령이 실패라는것을 의미하므로 then명령문다음의 명령문들은 무시되고 fi명령문다음에로 직접 조종이 넘어 간다.

조건지령들은 조립하여 설정할수 있다. 모든 if들은 대응되는 fi를 가져야 한다. fi는 if와 쌍을 이룬다. If블로크들의 형식을 사용하면 프로그람들의 오유수정을 쉽게 할수 있다.

형식

```
if 지령
then          #지령탈퇴상태의 검사
     지령
     지령
fi
- - - - - - - - - - - - - - - - - - - - - - - - - - -
if 검사표현식
then      # 표현식을 검사하기 위하여 검사지령을 사용
        지령
fi
                혹은
if  [ 표현식 ]
then                    # 낡은형의 검사지령을 사용
              지령      # 단어검사에 중괄호를 사용
fi
```

```
  - - - - - - - - - - - - - - - - - - - - - - - - - -
if  [[ 표현식 ]]
then              # 표현식을 검사하기 위하여 새로운 형의 중괄호를 사용
     지령
 fi
 - - - - - - - - - - - - - - - - - - - - - - - - - - -
 if 지령
 then
 . . .
 if 지령
 then
        . . .
 if 지령            # 겹쌓인 순환
 then
                . . .
                fi
         fi
  fi
```

실례 10-85

```
1.  if ypmatch $name passwd > /dev/null 2>&1
2.  then
echo Found $name!
3.  fi
```

설명

1. ypmatch지령은 봉사기의 NIS passwd자료기지에서 인수 name을 검사하는 NIS지령이다. 표준출력과 표준오유가 UNIX bit통인 /dev/null로 방향바꾸기 된다.
2. ypmatch지령의 탈퇴상태가 령이면 프로그람은 then명령문으로 가서 fi가 나타날 때까지 지령들을 실행한다.
3. fi는 then명령문다음의 지령목록을 끝낸다.

낡은 형의 Bourne test사용 사용자가 Bourne쉘에서 프로그람을 작성할 때 Korn쉘은 Bourne쉘스크립트를 원만히 실행하도록 허용하는 뒤방향일치성을 가진다. 많은 Bourne쉘프로그람작성자들은 Korn쉘로 변환할 때 여전히 표현식을 평가하기 위해 낡은 형의 검사지령을 사용한다. 사용자가 스크립트들을 읽거나 조작하면 응당 낡은 문법을 찾게 된다. 그러므로 새로운 Korn쉘 test지령으로 사용자자체의 스크립트를 작성하면 낡은 문법의 간단한 서술은 사용자를 도와 준다.

실례 10-86

```
#!/bin/ksh
```

```
    # Scriptname: are_you_ok
1.  print "Are you ok (y/n)   ?"
    read   answer
2.  if [ "$answer" =Y –o "$answer" = y ]          # Old-style test
    then
    print "Glad to hear it ."
3.  fi
```

설명

1. 사용자에게 《Are you ok(y/n)? 》라는 질문을 하고 read지령은 입력을 대기한다.
2. 중괄호 [에 의해 표현된 test지령은 표현식을 평가하기 위해 사용된다. 만약 표현식이 참이면 탈퇴상태 0을 되돌려 주고 표현식이 거짓이면 령이 아닌 값을 되돌려 준다. 변수 answer가 Y또는 y로 평가되면 then명령문다음의 지령들을 실행한다(test지령은 표현식을 평가할 때 통용기호사용을 허용하지 않는다.).
3. fi는 then명령문다음의 지령목록을 끝낸다.

새로운 형의 Korn test를 사용 새로운 Korn쉘형시험은 쉘메타문자들과 Korn쉘연산자 즉 &&와 || 들을 포함한 표현식을 허용한다.

실례 10-87

```
    #!/bin/ksh
    # Scriptname: are _you_ok2
1.  print "Are you ok (y/n) ?"
      read answer
2.  if [[ "$answer" = [Yy]* ]]               # New-style test
     then
         print "Glad to hear it ."
3.  fi
```

설명

1. 사용자에게 《Are you ok(y/n)?》라는 질문을 한다. read지령은 사용자입력을 대기한다.
2. 2중중괄호[[ ]]는 표현식을 평가하는데 사용하는 특수한 Korn쉘명령이다. 변수 answer가 Y 혹은 y다음에 여러개의 물음표로 평가되면 then명령문다음의 지령들이 실행된다.
3. fi명령문은 if를 끝낸다.

낡은 형 Bourne test와 수자의 사용 수값표현식을 검사하기 위해 낡은 형 Bourne쉘 test지령과 그의 연산자들을 그대로 Korn쉘에서 사용할수는 있지만 새로운 형의 let지령이 오히려 더 좋다.

실례 10-88

```
1.  if [ $#   -lt 1 ]
 then
        print   "$0: Insufficient arguments " 1>&2
        exit 1
2.  fi
```

설명

1. 명령문《인수들의 수가 1보다. 작으면 오유통보문을 현시하고 표준오유에 통보를 보내며 스크립트를 탈퇴한다.》을 읽는다. 낡은 형의 옹근수시험은 test 지령과 함께 사용된다.
2. fi는 then다음의 명령문블로크의 끝을 알려 준다.

let지령과 수자검사 물론 여전히 단일구분괄호들을 사용할수 있으며 낡은 형의 Bourne쉘연산자 더 좋기는 Korn쉘에서는 수값표현식을 검사할 때 2중소괄호와 C언어형의 값연산자를 사용한다. 2중소괄호는 오직 문자렬표현식시험와 파일시험에만 사용된다(표 10-29에서 보여 주었다.).

실례 10-89

```
1.  if((  $#  <  1))
    then
        print "$0: Insufficient arguments " 1>&2
        exit
2   fi
```

설명

1. 《인수들의 개수가 1보다. 작으면 오유통보를 출력하고 표준오유에 그것들을 보낸 다음 스크립트를 탈퇴한다.》라는 명령문을 읽는다. 이것은 Korn쉘에서 수값검사를 할 때보다 더 좋다.
2. fi는 then다음 명령문블로크의 끝을 알려 준다.

if/else지령 if/else지령은 두방향판정처리를 허용한다. if다음의 지령이 실패하면 else다음의 지령들이 실행된다.

형식

```
if 지령
then
     지령(들)
else
     지령(들)
fi
```

실례 10-90

```
1.  if ypmatch "$name " passwd>   /dev /null 2>&1
2.  then
            print Found $name!
3.  else
4.          print "Can't find $name ."
            exit 1
5.  fi
```

설명

1. ypmatch지령은 NIS passwd자료기지에서 인수 $name을 검색한다. 표준출력과 표준오유는 UNIX bit통인 /dev/null로 방향바꾸기 된다.
2. ypmatch지령의 탈퇴상태가 령이면 프로그람조종은 then명령문으로 이행하며 else가 나타날 때까지 지령들을 실행한다.
3. passwd자료기지안에서 ypmatch지령이 name을 찾지 못하면 else명령문다음 지령들이 실행된다. 즉 ypmatch의 탈퇴상태는 실행하려는 else블로크안의 지령들을 위해 령이 되지 말아야 한다.
4. print함수는 현시장치에 출력을 보내고 프로그람은 탈퇴한다.
5. 이 행은 if명령문의 끌을 알려 준다.

if/elif/else지령 if/elif/else지령은 여러 방향처리를 허용한다. if다음의 지령들이 실패하면 elif다음의 지령들이 검사된다. 지령이 성공하면 then명령문다음의 지령들이 실행된다. elif다음의 지령들이 실패하면 다음 elif지령이 검사된다. 그렇지 않으면 else지령들을 실행한다. else블로크는 기정이다.

형식

```
If 지령
then
    지령(들)
elif 지령
then
    지령(들)

elif 지령

then
    지령(들)
else
    지령(들)

fi
if  [[ 문자렬표현식 ]]      혹은      if (( 수값표현식))
then
    지령(들)
```

```
elif  [[  문자렬표현식  ]]  혹은   elif (( 수값표현식))
then
     지령(들)
elif  [[  문자렬표현식  ]]  혹은  elif (( 수값표현식))
then
     지령(들)
else
     자령(들)
fi
```

실례 10-91

```
(The  Script)
#!/bin/ksh
# Scriptname:  tellme
1.  read age?"How old are you? "
2.  if ((  age < 0 \ \  age > 120  ))
     then
            print "Welcome to our planet!! "
            exit 1
     fi
3.  if ((  age >= 0  &&  age < 13  ))
     then
          print "A child is a garden of verses"
          elif ((  age  >  12  &&  age  <  20  ))
     then

          print " Rebel without a cause"
         elif (( age >= 20 && age < 30 ))
     then
          print "You got the world by the tail!!"
          elif (( age >= 30 && age < 40 ))
     then

           print "Thirty something..."

4.  else
     print "Sorry I asked"
5.  fi
```

```
(The output)
$ tellme
How old are you?  200
Welcome to our planet!
```

```
$ tellme
How old are you?   13
Rebel without a cause

$ tellme
How are you?   55
 Sorry I asked
```

설명

1. 사용자에게 입력을 요구한다. 입력이 변수 age에 대입된다.
2. 수값시험은 2중소괄호안에서 실행된다. 변수 age가 0보다 작거나 120보다 크면 print지령이 실행되고 령이 아닌 탈퇴상태값으로 프로그람을 끝낸다. 대화형쉘재촉문이 나타난다. 2중소괄호(( ))연산자들을 사용할 때 화폐기호가 변수바꿔넣기에 필요 없다.
3. 수값검사가 2중소괄호안에서 실행된다. 변수 age가 0보다 크고 13보다 작으면 let지령은 참으로 탈퇴상태 0을 되돌려 준다.
4. else구성은 기정적이다. 아무런 명령문도 없으면 else명령문이 실행된다.
5. fi는 초기 if명령문을 끝낸다.

exit지령 exit지령은 스크립트를 끝내는데 사용하며 지령행으로 되돌려 준다. 사용자는 여러개의 조건이 참으로 검사되지 않으면 탈퇴하기 위해 스크립트를 요구할수 있다. exit지령의 인수는 0으로부터 255까지의 옹근수이다. 프로그람이 탈퇴할 때 탈퇴값은 쉘의 ?변수에 기억된다.

실례 10-92

```
(The script)
    #!/bin/ksh
    # Scriptname:   filecheck
    # Purpose: Check to see if a file exists,what type it is,
    # and its permissions.

1.  file=$1      # Variable is set to first command line argument

2.  if [[ ! –a $file ]]
    then
          print  " $file does not exist"
          exit 1
    fi
3.  if [[ -d $file ]]
    then
          print "$file is a directory"
4.  elif [[ -f $file ]]
```

```
            then
5.     if   [[ -r $file && -w $file && -x $file   ]]
  then
                    print"You have read,write,and   execute permission on
                    file $file"
     else
6.            print "You don't have the correct permissions"
              exit 2
            fi
    else
7.   print "$file is neither a file nor a directory. "
    exit 3
8.   fi
        (The Command Line)
9.   $ filecheck testing
    testing does not exist
10.   $ echo $?
    1
```

설명

1. 이 프로그람($1)을 의미하는 첫번째 지령행인수가 변수 file에 대입된다.
2. test지령은 if다음에 있다. $file이 존재하지 않는 (!부정연산자를 의미한다.)파일이면 then예약어 다음의 지령들이 실행된다. 령이 아닌 탈퇴값은 여러가지 원인(이 경우 검사가 실패하였기때문이다.)으로 프로그람이 실패한다는것을 의미한다.
3. 파일이 등록부이면 (file is a directory)를 현시한다.
4. 파일이 등록부가 아니면서 정확한 파일이면 then으로 간다.
5. 파일이 읽기가능, 쓰기가능 그리고 실행가능하면 then으로 간다.
6. fi는 제일 마지막 if지령을 끝낸다. 파일이 읽기, 쓰기 그리고 실행허가가 되여 있지 않으면 프로그람은 2인 인수를 가지고 탈퇴한다.
7. 2행과 3행이 실패하면 else지령들이 실행된다. 프로그람은 3이라는 값으로 탈퇴한다.
8. 실례에서fi는 3행에 있는 if로 이행한다.
9. testing이라는 파일이 존재하지 않는다.
10. 변수 $?는 령이 아닌 탈퇴상태값을 가진다.

null지령 null지령은 두점(:)을 의미한다. 이 지령은 탈퇴상태 0을 되돌려 주면서 아무런 동작도 하지 않는 내부지령이다. 이 지령은 아무런 동작도 요구하지 않을 때 if지령 다음에 써넣는다. 그렇지만 then명령문다음에 어떤 동작이 요구되거나 프로그람이 오유통보문을 처리하기 위해 필요하다. 보통 null지령은 [EDITOR:-/bin/vi]와 같은 변수표현식변경자들을 검사하거나 무환순환을 만들기 위해 loop지령의 인수로 사용한다.

실례 10-93

```
(The Script)
1.  name=Tom
2.  if grep "$name" databasefile >   /dev/null 2>&1
     then
3.       :
4.  else
              print "$1 not found in databasefile"
              exit 1
         fi
```

설명

1. 문자렬 Tom이 변수 name에 대입된다.
2. if지령은 grep지령의 탈퇴상태를 검사한다. Tom을 databasefile안에서 찾으면 null지령이 실행되며 아무런 동작도 하지 않는다.
3. 두점(:)은 null지령을 의미한다. 이 지령은 항상 탈퇴상태 0으로 탈퇴한다.
4. Tom을 찾지 못하면 실지 무엇을 하려고 한다는 오유통보가 현시되고 탈퇴한다. grep지령이 실패하면 else다음의 지령들이 실행된다.

실례 10-94

```
(The Script)
1.  : ${EDITOR:=/bin/vi}
2.  esho $EDITOR
```

설명

1. 두점(:)명령은 쉘에 의해 평가되는 인수를 가진다. 표현식 {EDITOR:=/bin/vi}은 두점명령에 대한 인수로 사용된다. 변수 EDITOR가 이미 설정되여 있으면 그 값은 변경되지 않는다. 만약 설정되지 않았으면 값/bin/vi가 그것에 대입된다. 표현식앞에 두점이 없으면Korn쉘은 《ksh:/bin/vi를 찾지 못한다.》와 같은 오유로 응답한다.
2. EDITOR의 값이 현시된다.

7. case지령

case지령은 if/else지령들가운데서 어느 하나를 선택하는 다중분기지령이다

case변수의 값은 대응되는 값을 찾을 때까지 value1, value2 순서로 대조한다.

값이 case변수와 일치하면 2중반두점에 도달할 때까지 값다음에 있는 지령들을 실행한다. 2중반두점이 아니면 esac가 나타날 때까지 *)다음에 있는 지령들을 실행한다. *)값은 if/else조건문에서 else명령문과 같은 목적으로 사용된다. case값들은 2개 값을 취급하는 수직막대기파이프기호와 쉘통용기호들을 사용할수 있다.

형식

```
case variable in
value1)
    command(s);;
value2)
    command(s);;
*)
    command(s);;
esac
```

실례 10-95

```
(The Script)
#!/bin/ksh
# Scriptname: xtermcolor
#Sets the xterm foreground color (the color of the prompt and
# input typed for interactive windows.
1.    read color?"Choose a foreground color for your terminal?"
2.    case "$color" in
3.    *[Bb]1??)
4.         xterm –fg blue –fn terminal &
5.                   ;;
6.     *[Gg]reen)
             xterm –fg darkgreen –fn terminal &
                 ;;
7      red \ orange)                    # The vertical bar means "or"
       xterm –fg "$color" –fn terminal &
              ;;
8      *) xterm –fn terminal &         # deault
             ;;
9.     esac
10.   print "Out of case…"
```

설명

1. 사용자에게 입력을 요구한다. 입력은 변수 color에 대입된다.
2. case명령은 표현식 $color를 평가한다.
3. color가 B 혹은 b로 시작하여 문자 1과 2개물음표들로 되여 있으면 case표현식에 첫번째 값을 대조한다. 값은 한개의 닫긴 소괄호에서 끝난다. 통용기호는 쉘메타문자이다.
4. 3행의 값이 case표현식에 대응되면 명령문을 실행한다. xterm지령은 전면색을 푸른색으로 설정한다.
5. 2중반두점은 이 지령블로크에서 지령 다음에 있어야 한다. 반두점에 도달하

면 조종은 10행으로 이행한다.

6. 만일 case표현식이 G 혹은 g다음에 r-e-e-n문자들에 대응되면 xterm창문의 전면색은 어두운 풀색으로 설정된다. 2중반두점은 명령문들의 블로크를 끝맺으며 조종은 10행으로 이행한다.
7. 수직막대기는 합조건연산식으로 사용된다. 만일 case표현식이 red나 drange로 되면 term지령이 실행된다.
8. 이것은 기정값이다. 만일 case표현식에 대응되는 값이 아무도 없으면 *)다음의 지령들이 실행된다. 말단 전면의 기정색은 검은색이다.
9. esac명령문은 case지령을 끌낸다.
10. 값들중 하나가 선택된 다음에 여기서 실행이 계속된다.

case명령과 here문서 일반적으로 here문서는 차림표를 작성하기 위해 사용한다. 사용자가 차림표로부터 요구되는것을 선택한 다음 case지령은 요구되는것들가운데서 한개를 다시 대조한다. Korn쉘은 또한 차림표를 작성하기 위해 select순환을 제공한다.

실례 10-96

```
(The .profile File)
     print   "Select a terminal type "
1.   cat << EOF
                 1) vt120
                 2) wyse50
                 3) ansi
                 4) sun
2.   EOF
3.   read TERM
4.   Case "$TERM" in
      1)   export TERM=vt120
           ;;
      2)   export TERM=wyse50
           ;;
      3)   export TERM=ansi
           ;;
      *)   export TERM=sun
           ;;
5.   esac
     print "TERM is $TERM"
```

설명

1. here문서는 요구되는 차림표를 현시하기 위해 사용한다.
2. EOF는 사용자정의 끝지적자이다. here문서에 대한 입력은 여기서 중지된다.

3. read지령은 입력을 기다리며 변수TERM에 그것을 대입한다.
4. case지령은 변수 TERM을 평가하고 목록에서 그중 한개의 수값을 다시 대조한다. 만일 선택이 끝나면 설정을 끝낸다.
5. case명령은 esac에서 끝난다.

8. 순환지령.

순환지령들은 조건이 성립될 때까지 지령이나 지령그룹를 반복하여 실행하기 위해 사용된다. Korn 쉘에는 4개의 순환 for loop, while loop, until loop 그리고 select loop가 있다.

for지령 for순환지령은 설정된 인수들가운데서 매개 원소들에 해당한 지령들을 실행하기 위해 사용된다. 파일이나 사용자이름목록에서 똑 같은 지령들을 실행하기 위해 이 순환을 사용한다. for지령 다음에는 사용자정의변수가 있고 그다음 예약어 in 그리고 단어목록이 있다. 순환에서는 먼저 단어목록으로부터 한개 단어씩 변수에 계속 대입한다. 순환본체는 do예약어로 시작하여 done예약어에서 끝난다. 목록안에 있는 모든 단어들이 밀기되면 순환은 끝나고 프로그람조종은 done예약어 다음에로 넘어 간다.

형식

```
for  변수 in 단어목록
do
        지령(들)
done
```

실례 10-97

(The Script)

1. **for pal in Tom Dick Harry Joe**
2. **do**
3. print “Hi $pa1”
4. **done**
5. print “Out of loop”

(The Output)

Hi Tom
Hi Dick
Hi Harry
Hi Joe
Out of loop

설명

1. 이 for순환은 목록이름 즉 Dick, Harry 그리고 Joe를 가지고 변수 pal에 대입한 다음 왼쪽으로 하나씩 밀면서 반복실행한다. 따라서 모든 단어들이 밀기되면 단어목록은 비게 되여 순환은 끝나고 done예약어다음의것이 실행되기

시작한다. for지령다음의 단어 pal은 순환의 매 단계에서 in예약어다음의 값이 대입되는 변수이다. 처음 변수 pal에 단어 Tom을 대입한다. 두번째로 pal에 Dick를 대입하며 다음에 Harry를 대입하고 마지막에 Joe를 대입한다.

2. do예약어는 단어목록다음에 있어야 한다. 만일 같은 행우에서 사용하려면 목록이 반두점으로 끝나야 한다. 실례로 for pal in Tom Dick harry Joe ; do이다.
3. 이것은 순환본체이다. Tom이 변수 pal에 대입된 다음 순환본체에 있는 지령들 즉 do와 done예약어사이에 있는 모든 지령들이 실행된다.
4. done예약어는 순환을 끝낸다. 만일 1행의 단어목록에서 처리하려는 단어들이 없으면 순환은 탈퇴하고 5행을 실행하기 시작한다.
5. 이 행은 순환이 끝날 때 실행된다.

실례 10-98

(The Command Line)

```
1.  $ cat mylist
tom
partty
ann
jake
```

(The Script)

```
2.  for person in $( <   mylist)       #same as for person in   'cat mylist'
     do
3.                 mail $person < letter
            print $person was sent a letter.
4.  done
5.  print "The letter has been sent."
```

설명

1. mylist라는 파일의 내용이 현시된다.
2. 지령바꿔넣기가 실현되고 mylist의 내용이 단어목록으로 된다. 순환에서는 우선 tom이 변수 person에 대입된 다음 patty등으로 치환하기 위해 밀기된다.
3. 순환본체에서 letter라는 파일의 복사를 우편으로 전송한다.
4. done예약어는 순환끝을 식별하는 표식이다.
5. 목록에 있는 모든 사용자들이 전자우편을 전송하였을 때 순환이 끝나므로 이 행이 실행된다.

실례 10-99

```
1.  for file   in *.c
```

```
2.  do
          if [[ -f   $file ]]   ; then
          cc $file   -o ${fiel%.c}
        fi
    done
```

설명

1. 단어목록은 확장자가 .c(c원천파일들)로 끝나는 현재작업등록부의 모든 파일들로 된다. 매 파일이름들은 순환의 매 단계마다 변수 file에 대입된다.
2. 순환본체가 입력될 때 파일은 그것이 존재하는가와 실지 파일인가를 검사한다. 그렇다면 그것은 번역될것이다. ${file%.c}는 c라는 확장자로 파일이름을 확장한다.

단어목록에서 $*와 $@변수 변수를 확장할 때 2중인용부호안에 있지 않으면 $*와 $@는 서로 같다. “$*”는 한개의 문자렬을 평가하고 “$@”는 분리된 단어목록을 평가한다.

실례 10-100

```
(The Script)
    #!/bin/ksh
1.     for name in $*        # or for name in $@
2.     do
           echo Hi $name
3.   done
 - - - - - - - - - - - - - - - - - - - - - - - - -
(The Command Line)
$ greet Dee Bert Lizzy Tommy
Hi Dee
Hi Bert
Hi Lizzy
Hi Tommy
```

설명

1. $*와 $@는 모든 위치파라메터목록을 확장한다. 이 경우 인수들은 지령행으로부터 넘겨진다. 즉 Dee, Bert, Lizzy 그리고 Tommy이다. 목록안의 매개 이름은 for순환에서 변수 name에 대입된다.
2. 순환본체에 있는 지령들은 목록이 빌 때까지 실행된다.
3. done예약어는 순환본체의 끝을 알려 준다.

while지령 while은 자기다음에 있는 지령을 즉시 평가하여 탈퇴상태가 령이면 순환본체안에 있는 지령(do와 done사이에 있는 지령들)들을 실행한다.

done예약어에 도달하면 순환의 첫 시작으로 조종을 되돌리며 while지령은 지령의

탈퇴상태를 다시 검사한다. 탈퇴상태가 0이 아니면 프로그람은 done예약어다음의것을 실행하기 시작한다. 지령의 탈퇴상태가 0아닌 값이 나타날 때까지 순환은 계속된다.

지령의 탈퇴상태가 0이 아니면 프로그람은 done예약어다음의것을 실행한다.

탈퇴상태가 0으로만 되면 무환순환에 들어 간다(물론 Ctrl-C 혹은 Ctrl-\ 처리로 순환을 중지시킬수 있다.).

형식

```
while 지령
do
        지령(들)
done
```

실례 10-101

(The Script)

```
1.  num=0                                   # Initialize num
2.  while (( num < 10 ))                    # Test num with the let
    do
        print -n $num
3.      (( num=num + 1 ))                   # Increment num
    done
    print "\ nAfter loop exits, continue running here"
```

(The Output)

0123456789

After loop exits, continue running here

설명

1. 이것은 초기화단계이다. 변수 num에 0을 대입한다.
2. while지령다음에 let지령이 있다. num값이 10보다 작으면 순환본체에 들어 간다.
3. 순환본체에서 num값이 하나씩 증가한다. 만일 num값이 변하지 않았으면 순환은 처리가 중지될 때까지 계속 반복한다.

실례 10-102

(The Script)

```
    #!/bin/ksh
    # Scriptname: quiz
1.  read answer ?"Who was the U.S. President in 1992? "
2.  while [[ $answer != "Bush" ]]
3.  do
```

print “ Wrong try agein!”
4. **read answer**
5. **done**
6. print Good guess!

(The Output)
$ **quiz**
Who was the U.S. President in 1992? ***George***
Wrong try again!
Who was the U.S. President in 1992? ***I give up***
Wrong try again!
Who was the U,S. president in 1992? ***Bush***
Good guess! .

설명

1. read지령은 물음표(?)다음에 있는 문자렬 즉《who the U.S president in 1992?》을 출력하고 사용자입력을 기다린다.
2. while순환에 들어 가면 변수 answer에 기억된다.
   test지령 중괄호 [[는 표현식들을 평가한다. 변수 answer가 문자렬 brath와 같지 않다면 순환본체에 들어 가고 do와 done사이에 있는 지령들을 실행한다.
3. do예약어는 순환본체의 시작이다.
4. 사용자에게 다시 입력할것을 요구한다.
5. done예약어는 loop본체의 끝을 알려 준다. 조종은 while순환의 시작으로 되돌아 가며 표현식이 다시 검사한다. $answer가 문자렬 “bash”이면 순환은 끝나고 프로그람조종은 6행으로 이행 한다.

실례 10-103

(The Script)
1. **go=1**
   print Type q to quit.
2. **while let go** *or* **((go))**
   do
   print I love you.
   read word
3. **if [[$word = [qo]*]]**
   then
   print “I’ll allways love you”
4. **go=0**
   fi

5. done

(The Output)
$ **sayit**
Type q to quit.
I love you
I love you.
I love you.
I love you.
I love you.
q
I ’ll allways love you.
$

설명

1. 변수 go에 1을 대입한다.
2. 순환에 들어 간다. let지령은 표현식을 검사한다. 표현식들을 하나로 평가한다. 프로그람이 while순환안으로 들어 가며 do예약어로부터 done예약어까지의 명령들을 실행한다.
3. 사용자가 변수 word에 Q또는 q를 입력하면 then과 fi사이에 있는 명령들이 실행된다. 그렇지 않으면 《I love you》가 현시된다.
4. 변수 go에 0을 대입한다. 프로그람조종이 while순환의 머리부에서 시작할 때 식이 검사된다. 그때 표현식은 거짓으로 평가되므로 순환에서 탈퇴되고 5행에 있는 done예약어다음의 스크립트가 실행을 시작한다.
5. done예약어는 순환본체의 끝을 알려 준다.

until지령 until지령은 while지령과 비슷하지만 반대방향의 탈퇴상태를 평가한다. 명령을 직접 평가하고 탈퇴상태가 령이 아니면 순환본체안에 있는 지령(do와 done사이에 있는 지령들)들을 실행한다. done예약어에 도달하면 순환의 시작으로 되돌아 가며 until지령은 지령의 탈퇴상태를 다시 검사한다. 지령의 탈퇴상태가 0으로 될 때까지 순환을 계속 반복한다. 탈퇴상태가 0으로 될 때 done예약어다음에 있는 프로그람이 실행되기 시작한다.

형식

```
until  지령
do
     지령
done
```

실례 10-104

#!bin/ksh
1. **until who \ grep limda**

```
2.  do
     sleep 5
3.  done
    talk linda@dragonwings
```

설명

1. until순환은 파이프행의 마지막지령 grep의 탈퇴상태를 검사한다. who지령은 이 기계에 누가 등록가입하는가를 현시하며 grep지령에로 출력을 파이프한다. grep지령은 사용자 linda를 찾았을 때에만 탈퇴상태 0(성공)을 되돌려 준다.
2. 만일 사용자 linda가 등록가입되지 않을 때 순환본체에 들어 가고 프로그람은 5s동안 대기한다.
3. 사용자 linda가 등록가입될 때 grep지령의 탈퇴상태는 0이 아니며 조종은 done예약어다음의 명령으로 이행한다.

실례 10-105

```
    #!/bin/ksh
1.  hour=0
2.  unitl (( hour > 23 ))
    do
3.      case  "$hour "  in
         [0-9]|1[0-1])  print  "Good moning!"
              ;;
        12) print  "Lunch time"
              ;;
        1[3-7])  print "siesta time"
              ;;
        *)  print "Good nigtht"
             ;;
        esac
4.      (( hour+=1 ))
5.  done
```

설명

1. hour변수에 0을 대입한다. 변수는 until순환에서 사용되기전에 초기화되여야 한다.
2. until지령다음에는 let지령이 있다. hour가 23보다 크지 않으면 탈퇴상태는 0이 아니며 순환본체로 들어 간다.
3. case지령은 hour값중의 한개를 다시 hour변수값으로 대응시키거나 혹은 기

정값으로 대응시켜 응용된 지령을 실행한다.

4. hour를 하나 증가시킨다. 만약 hour가 23보다 크지 않으면 순환은 탈퇴하지 않는다. 조종은 until지령으로 되돌아 가서 hour를 다시 평가한다.
5. done예약어는 순환의 끝을 알리는 표식이다. hour가 23보다 크면 done 다음행으로 조종을 넘긴다.

select지령과 차림표 here문서는 쉽게 차림표를 작성하기 위한 방법이지만 Korn 쉘은 본래부터 차림표를 작성하는데 사용되는 select순환이라는 새로운 순환을 소개한다. 수값으로 목록화된 항목들로 구성되는 차림표는 표준오유에 현시된다. PS3재촉문 즉 #?는 기정으로 사용자입력을 대기하는데 사용된다. PS3재촉문이 현시된 다음 쉘은 사용자입력을 대기한다. 입력은 차림표목록안의 수값들가운데서 한개를 설정해야 한다.

입력은 특수한 Korn 쉘 REPLY변수에 기억된다. REPLY변수의 값에는 목록선택에서 괄호오른쪽에 있는 문자렬이 결합된다.[7] case지령은 사용자의 요구를 차림표로부터 선택하기 위해 select지령과 함께 사용되며 선택에 기초하여 해당한 지령들을 실행한다. LINES와 COLLEMS변수들은 말단에 현시되는 차림표항목들의 배치상태를 확정하기 위해 사용할수 있다. 출력은 표준오유에 현시되며 매 항목다음에는 수자와 닫기괄호가 붙으며 PS3재촉문이 차림표아래에 현시된다. select지령이 순환지령이기때문에 순환을 벗어 나기 위한 break지령과 스크립트를 탈퇴하기 위한 exit지령을 사용하는것이 중요하다.

형식

```
select var in wordlist
do
    command(s)
done
```

실례 10-106

(The Script)

```
        #i/bin/ksh
# Program name: goodboys
1.  PS3="Please choose one of the three boys : "
2.  select choice in tom dan guy
3.  do
4.        case $ choice in
           tom)
      print Tom is a cool dude!
5.                 break;;           # break out of the select loop
6.        dan | guy )
              print Dan and Guy are both sweethearts.
```

7. 순환이 다시 시작할 때 차례표를 다시 현시하려면 done 예약어앞의 빈 자리에 REPLY 변수를 설정해야 한다.

```
            break;;
                  *)
7.                print " $REPLY is not one of your choices" 1>&2
                    print "Try again."
                    ;;
8.          esac
9.   done
```

(The Command Line)
$ **goodboys**
1) tom
2) dan
3) guy
Please choose one of the three boys : ***2***
Dan and Guy are both sweethearts.

$ **goodboys**
1) tom
2} dan
3) guy
Please choose one of the three boys : ***4***
4 is not one of your choices
Try again.
Please choose one of the three boys : ***1***
Tom is a cool dude !
$

설명

1. PS3변수에는 차림표선택목록아래 나타나는 재촉문을 대입한다. 재촉문이 현시된 다음 프로그람은 사용자입력을 대기한다. 입력은 REPLY라는 내부변수에 기억된다.
2. select지령다음에는 변수 choice가 있다. 문법은 for순환과 비슷하다. 변수 choice에는 다음에 있는 목록가운데서 매 항목들을 차례로 대입한다. 그것들은 tom, dan 그리고 guy이다. 이것은 수값과 오른쪽 소괄호가 앞에 있는 차림표에서 현시되는 단어목록이다.
3. do예약어는 순환의 시작을 가리킨다.
4. select순환의 첫번째 지령은 case지령이다. case지령은 보통 select순환과 함께 사용된다. REPLY변수의 값에는 선택하려는것들중에서 한개가 련결된다. 즉 1은 tom에 련결되고 2는 dan에 련결되며 3은 guy에 련결된다.
5. Tom이 선택되면 tom is cool dude!를 현시한 다음 break지령이 select순환에서 탈퇴하도록 한다. 출력된 다음 프로그람조종에 의하여 done예약어 다음에로 이동한다.
6. 2(dan)와 3(tom) 차림표항목이 설정되면 REPLY변수는 사용자가 선택한것

을 가지게 된다. 설정이 1, 2, 3이 아니면 오유통보를 표준오유에 보낸다. 사용자에게 다시 설정하겠는가를 물어 본 다음 조종은 select순환의 시작으로 넘어 간다.

7. case지령의 끝이다.
8. select순환의 끝이다.

실례 10-107

The Script)

```
    #!/bin/ksh
    # Program name: ttype
    # Purpose: set the terminal type
    #Author: Andy Admin
1.  COLUMNS=60
2.  LINES=1
3.  PS3="Plase enter the terminal type: “
4.  select choice in wyse50 vt200 vtlOO sun
    do
5.      case $REPLY in
         1)
6.          export TERM=$choice
            print "TEKM=$choice"
            break;;                    # break out of the select loop
         2 | 3 )
            export TERM=$choice
            print "TERM=$choice"

            break ; ;
         4 )
            export TERM=$choice
            print " TERM=$choice"
            break;;

          *)

7.          print "$REPLY is not a valid choice. Try again" 1>&2
                ;;
          esac
8.  done
```

(The Command Line)

```
$ ttype
1) wyse50     2) vt200    3) vtlOO     4) sun
Please enter the terminal type : 4
```

TERM=sun

$ **ttype**
1) wyse5O 2) vt200 3) vt1OO 4) sun
*Please enter the terminal type : **3***
TERM=vtlOO

$ **ttype**
1) wyse5O 2) vt200 3) vtlOO 4) sun
*Please enter the terminal type : **7***
7 is not a valid choice. Try again.
*Please enter the terminal type: **2***
TERM=vt200

설명

1. COLUMNS변수는 select순환에서 작성된 차림표의 렬에 말단현시의 너비값을 설정한다. 기정값은 80이다.
2. LINES변수는 말단우에서 select차림표의 수직현시를 조종한다. 기정값은 24행이다. 마지막실례에서와 같이 LINES값을 1로 변경하면 베이치도표대신에 차림표항목들은 한개 행우에 현시된다.
3. PS3재촉문이 설정되고 차림표option아래에 현시된다.
4. select순환은 4개의 option 즉 wyse50, vt 200 그리고 sun으로 된 차림표를 현시한다. 변수 choice에는 REPLY변수에 기억되여 있는 사용자응답에 해당한 한개 값이 대입된다. 만약 REPLY가 2이면 choice에는 vt 200이 대입되며 REPLY가 3이면 choice에는 vt 100을 대입하며 REPLY가 4이면 choise에는 sun이 대입된다.
5. REPLY변수는 사용자의 입력선택을 평가한다.
6. 말단형이 대입되고 반출되며 출력된다.
   만일 사용자가 1과 4사이의 값을 입력하지 않으면 다시 입력할것을 재촉한다. 차림표가 나타나지 않고 PS3이 나타난다. 차림표를 다시 나타내기 위해서는 8행에서 REPLY변수를 null로 해야 한다.
7. select순환의 끝이다.

순환지령 여러가지 조건에 따라 순환밖으로 탈퇴하거나 순환머리부로 되돌아 가거나 순환을 정지할수 있다. Korn쉘은 순환을 조종하기 위한 조종지령들을 제공한다.

shift지령 shift지령은 왼쪽으로 임의의 수만큼 파라메터목록을 밀기한다. 인수가 없는 shift지령은 왼쪽으로 오직 한개 파라메터목록을 밀기한다. 목록이 밀기되면 파라메터는 아주 소거된다. shift지령은 위치파라메터들의 목록조종을 위해 while순환에서 자주 사용된다.

형식

```
shift [n]
```

실례 10-108

(Without a Loop)
(The Script)

```
    #!/bin/ksh
    # Scriptname: doit 0

1.  set joe mary tom sam
2.  shift
3.  print $*
4.  set $(date)
5.  print $*
6.  shift 5
7.  print $*
8.  shift 2
```

(The Output)

```
    $ doit0
3.  mary tom sam
5.  Sun Sep 9 10:00:12 PDT 2001
7.  2001
8.  ksh: shift: bad number
```

설명

1. set지령은 위치파라메터를 설정한다. $1에는 joe가 대입되고 $2에는 mery가 대입되며 $3에는 tom이 대입되고 $4에는 sam이 대입된다.
2. shift지령은 왼쪽으로 위치파라메터를 밀기한다. 그러면 joe가 밀기된다.
3. 파라메터목록이 밀기된 다음 현시된다. $*는 모든 파라메터들을 의미한다.
4. set지령은 UNIX date지령의 출력으로 위치파라메터들을 재설정한다.
5. 새 파라메터목록이 출력된다.
6. 이때 목록은 왼쪽으로 5번 밀기된다.
7. 새 파라메터목록이 출력된다.
8. 소괄호안에 있는 파라메터보다 더 많이 밀기하려고 하면 쉘은 표준오유에 통보를 보낸다.

실례 10-109

(With a Loop)
(The Script)

```
    #!/bin/ksh
    # Usage: doit [args]
1.  while (( $# > 0 ))
```

```
      do
2.        print $*
3.        shift
4.    done
```

(The Command Line)

```
$ doit  a b c d e
a b c d e
b c d e
c d e
d e
   e
```

설명

1. while지령은 수값표현식을 검사한다. 만일 위치파라메터의 개수($#)가 0보다 크면 순환본체에 들어 간다. 위치파라메터들은 인수로서 지령행으로부터 넘겨 진다. 모두 5개이다.
2. 모든 위치파라메터들이 출력된다.
3. 파라메터목록이 왼쪽으로만 밀기된다.
4. 순환본체가 여기서 끝난다. 조종은 순환의 머리부에로 이행한다. 파라메터목록이 하나씩 감소된다. 첫번째 밀기한 다음의 $#는 4이다. $#가 0으로 감소될 때 순환은 끝난다.

break지령 내부지령 break는 순환으로부터 즉시 탈퇴하려고 할 때 사용되지만 프로그람으로부터 탈퇴하려고 할 때에는 사용하지 않는다(프로그람을 끝내기 위해서는 exit지령을 사용해야 한다.). break지령이 실행된 다음에 조종은 done예약어다음으로 이행한다. break지령은 제일 안쪽에 있는 순환으로부터 탈퇴하게 하므로 다중순환에서는 어느 순환에서 탈퇴하는가를 지적하는 인수값을 가진다. break지령은 무한순환으로부터 탈퇴하기 위해 사용된다.

형식

```
break [n]
```

실례 10-110

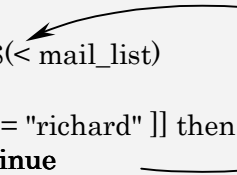
```
1.  while true;  do
2.        read answer? Are you ready to move on\ ?
3.        if [[ $answer = [Yy]* ]]; then
                  break
4.        else
                    .... commands...
              fi
5.  done
6.  print "Here we are"
```

설명

1. true지령은 UNIX지령이며 korn쉘에서 두점(:)지령을 위한 별명이다. 그것은 항상 0상태로 탈퇴하며 무한순환을 시작하기 위해 자주 사용된다(null지령 (:)은 똑 같은 동작을 하려고 할 때 사용할수 있다.). 순환본체에 들어 간다.
2. 사용자에게 입력을 요구한다. 사용자입력이 변수 answer에 대입된다.
3. answer가 y, Y, Yes, Yup 혹은 Ya(Y또는 y로 시작하는 모든것)이면 break지령이 실행되고 조종은 6행으로 넘어 간다. 《Here we are》가 출력된다. 사용자가 Y혹은 y로 시작하는 문자렬로만 대답한다면 프로그람은 입력요구를 반복한다. 이것은 무조건이행이다.
4. 검사가 3행에서 실패하면 else지령이 실행된다. 순환본체가 done예약어에서 끝날 때 조종은 1행에 있는 while의 머리부에로 다시 이행한다.
5. 순환본체의 끝이다.
6. 조종은 break지령이 실행된 다음에 여기서부터 시작한다.

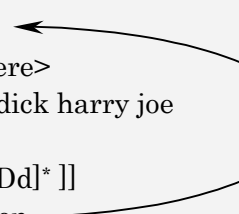
continue지령 continue지령은 일부조건이 참이면 머리부에서 다시 시작한다. continue아래에 있는 모든 지령들을 무시한다. continue지령은 제일 마지막순환의 머리부에로 조종을 되돌린다. 즉 순환이 수값을 가지고 설치되여 있으면 continue지령은 인수로 그 값을 가져야 한다. 조종은 임의의 어떤 번호를 가진 순환의 첫 시작위치에로 이행한다.

형식

continue [값]

실례 10-111

(The Mailing List)
$ cat mail_list
ernie
john
richard
melanie
greg
robin

(The Script)
 # Scriptname: mailtomaillist

```
#! /bin/ksh
1.   for name in $(< mail_list)
do
2.   if [[ "$name" = "richard" ]] then
3.             continue
          else
4.             mail $name < memo
          fi
5.   done
```

설명

1. for순환은 mail_list라고 불리우는 파일에 기억된 목록이름을 가지고 순환한다. 매번 목록으로부터 이름이 변수 name에 대입되고 목록이 밀기되며 다른 이름으로 바꿔넣기된다.
2. name에 richard가 대응되며 continue지령이 실행된다. 그때 richard가 밀기되기때문에 다음 사용자 즉 melanie가 변수 name에 대입된다.
3. continue지령은 순환의 머리부로 조종을 되돌리며 순환본체의 나머지 지령들은 실행되지 않는다.
4. 목록에서 richard를 제외한 모든 사용자들은 파일 memo의 복사를 우편으로 전송된다.
5. 순환본체의 끝이다.

겹쌓인 순환 겹쌓인 순환에서 break지령과 continue지령은 순환을 끝내는 조종을 할수 있다.

실례 10-112

```
(The Script)
     #!/bin/ksh
1.   while true ; do
     < Commands here>
2.   for user in tom dick harry joe
             do
        if [[ $user = [Dd]* ]]
                    then
3.               continue 2
           < commands here >
4.                          while true
                              do
              < commands here>
5.                               break   3
6.                          done
7.            fi
          done
8.   done
9.   print Out of loop
```

설명

1. true지령은 항상 탈퇴상태 0을 되돌려 준다. 순환은 loop조종명령을 사용하지 않는한 무한순환으로 된다.
2. for순환에 들어 간다.

3. for지령은 목록에 있는 매개 이름에 의해 순환된다. 만일 사용자변수가 D 혹은 d이면 continue지령은 while순환의 머리부로 조종을 넘긴다. 인수가 없으면 continue지령은 for순환의 머리부에서부터 조종을 시작한다. 인수 2는 두번째 닫긴 순환의 머리부로 가도록 쉘을 가리키며 그것에서부터 다시 실행되기 시작한다.
4. while순환이 설치된다. True지령은 항상 상태값 0을 가지고 탈퇴한다. 무한순환으로 된다.
5. break지령은 제일 마지막 while순환을 끌낸다. 9행을 실행하기 시작한다.
6. done예약어는 제일 안쪽 while순환의 끝을 알려 준다.
7. 이 done예약어는 for순환의 끝을 알려 준다.
8. 이 done은 제일 바깥쪽 while순환의 끝을 현시한다.
9. 순환의 바깥이다.

I/O방향바꾸기와 순환 korn쉘에서는 순환에서 방향바꾸기와 파이프를 사용할수 있다. Bourne쉘과는 달리 순환은 자식쉘이 아니라 이 쉘안에서 집행된다. 순환에서 설정된 변수들은 순환이 탈퇴할 때까지 설정된다.

순환의 출력을 파일로 방향바꾸기 순환의 출력을 현시장치로 보낼 대신에 파일이나 파이프로 방향바꾸기할수 있다. 실례 10-113에서 보여 준다.

실례 10-113

```
 (The command Line)
1.   $ cat memo
     abc
     def
     ghi
--------------------------------------------------------------------------------
(The Script)
     #!/bin/ksh
     # Program name: numberit

     # Put line numbers   on all lines of memo
2.     if (( $# < 1 ))
          then
          print "Usage: $0 filename " >&2
          exit 1
       fi
3.   integer count= 1                    # Initialize count
4.   cat $1 | while read line            # Input is coming from memo
            do
5.      (( count = =1 )) &&   print "Processing file $1..." > /dev/tty
6.      print $count $line
7.      (( count+=l ))
8.   done > tmp$$                        # Output is going to a temporary file
9.   mv tmp$$ $1
```

(The Command Line)
10. $ **number it memo**
Processing file memo...

11. $ **cat memo**
1 abc
2 def
3 ghi

설명

1. 파일memo의 내용이 현시된다.
2. 만일 인수의 개수가 1보다 작으면 표준오유인 현시장치에 취급통보문을 보낸다.
3. count변수는 옹근수로 정의되고 1을 대입한다.
4. UNIX cat지령은 $1에 기억된 파일이름의 내용을 현시하며 출력은 while순환으로 파이프된다. read지령에는 순환의 첫 단계에서 파일안에 있는 첫번째 행이 대입되고 다음단계에는 두번째 행이 대입되는 식으로 대입된다.
5. 이 print명령문의 출력은 현시장치 즉 /dev/tty에로 보내진다. Dev/tty로 정확히 방향바꾸기를 하지 않으면 출력은 8행에 있는 tmp$$로 방향바꾸기된다.
6. print함수는 count값다음에 파일안에 있는 행을 현시한다.
7. count변수가 하나 증가한다.
8. 3행을 제외한 입력순환의 출력은 파일 tmp$$로 방향바꾸기 된다($$는 이 프로쎄스의 PID를 평가한다.). tmp파일에는 이 프로쎄스의 PID이름에 첨가한 단 한개의 이름만을 준다.
9. tmp파일은 $1에 대입되여 있는 파일이름으로 다시 이름 지어 진다.
10. 프로그람이 실행되며 처리하기 위한 파일은 memo로 된다.
11. 파일은 행번호와 함께 현시된다.

UNIX지령에로 순환의 출력을 파이프하기 순환의 출력은 현시장치로부터 파이프에로 방향바꾸기할수 있다. 실례 10-114에서 보여 준다.

실례 10-114

(The Script)
1. for i in 7 9 2 3 4 5
2. do
        print $i
3. **done | sort -n**

(The Output)
2
3
4
5
7
9

설명

1. for지령은 분류되지 않는 수자목록을 가지고 순환한다.
2. 순환본체에서 번호들이 현시된다. 이 출력은 UNIX sort지령에 파이프된다.
3. 파이프는 done예약어다음에 작성된다.

배경에서 순환의 실행 순환에서 프로쎄스를 계속 실행하려면 배경일감으로 처리하여 프로그람의 나머지 부분을 계속 실행하여야 한다.

실례 10-115

```
1.  for person in bob jim joe sam
        do

2.  mail $person < memo
3.  done &
```

설명

1. for순환은 단어목록 즉 bob, jim, joe 그리고 sam에서 매개 이름들을 밀기한다. 매개 이름들이 변수 person에 대입된다.
2. 순환본체의 매개 person에 파일 memo의 내용을 보낸다.
3. done예약어의 끝에 준것은 배경에서 실행되도록 순환을 조작하기 위한것이다. 프로그람은 순환이 실행될 때에도 집행을 계속한다.

exec지령과 순환 exec지령은 자식쉘을 작성함이 없이 표준입력, 표준출력을 닫는데 사용할수 있다.

실례 10-116

```
(The File)
1.  cat tmp
     apples
     pears
     bananas
     peaches
     plums
-----------------------------------------------------
(The Script)
     #!/bin/ksh
     # Scriptname: speller
     # Purpose: Check and fix spelling errors in a file
     #

2.  exec < tmp              #Opens the tmp file
3.  while read line         #Read from the tmp file
     do
4.       print $line
5.       print –n “Is this word correct? [Y/N] “
```

```
6.      read answer < /dev/tty        # Read form the terminal
          case $answer in
            [Yy]*)
          continue
             ;;
               *)
                          print "New word "
7.      read   word < /dev/tty
         sed "s/$line/$word/" tmp > error
          mv error   tmp
 8.       print   $word has been changed
                    ;;
             esac
          done
```

설명

1. tmp파일의 내용이 현시된다.
2. exec지령은 표준입력(파일서술자 0)을 변경하므로 건반으로 입력할 대신 tmp파일로부터 입력된다.
3. while순환이 시작된다. read지령은 tmp파일로부터 입력행을 얻는다.
4. 변수 line에 의해 기억된 값이 현시장치에 현시된다.
5. 단어가 정확한가를 사용자에게 물어 본다.
6. read지령은 말단 즉 /dev/tty로부터 사용자의 응답을 얻는다. 입력이 말단으로 즉시 방향이 바꾸어 지지 않으면 입력이 열릴 때까지 tmp파일로부터 읽기는 계속된다.
7. 사용자에게 다시 입력을 요구하며 입력이 말단 /dev/tty로 방향바꾸기된다.
8. 새 단어가 현시된다.

IFS와 순환 IFS 즉 쉘의 내부마당식별자는 공백, 타브, 행바꾸기문자들을 평가한다. 그것은 read, set, for, select와 같은 단어목록들을 해석하는 지령의 단어식별자로 사용된다. 여러가지 식별자들이 목록에서 사용된다면 사용자에 의하여 재설정할수 있다. 그것은 변경하기전에 다른 변수에 IFS의 원래의 값을 기억하기 위한 좋은 방법이다. 또한 기정값으로 되돌아 가기는 쉽다.

실례 10-117

```
(The Script)
    #!/bin/ksh
    # Script is called runit.
    # IFS is the internal field separator and defaults to
    # spaces, tabs, and newlines.
    # In this script it is changed to a colon.
1.  names=Tom:Dick:Harry:John
2.  OLDIFS="$IFS"                # Save the original value of IFS
3.  IFS=" : "
4.  for persons in $names
```

```
        do
5.          print Hi $persons
           done
6.     IFS="$OLDIFS"                # Resert the IFS to old value
7.     set Jill Jane Jolene         # Set positional parameters
8.     for girl in $*
          do
          print Howdy $girl
          done
```

(The output)

```
$    runit
Hi Tom
Hi Dick
Hi Harry
Hi John
Howdy Jill
Howdy Jane
Howdy Jolene
```

설명

1. names변수에 문자렬 Tom:Dick:Harry:John을 설정한다. 매 단어는 옹근두점으로 구별된다.
2. IFS의 값을 다른 변수 OLDIFS에 대입한다. IFS의 값이 공백이기때문에 공백을 유지하기 위해 인용부호안에 넣어야 한다.
3. IFS에 두점(:)을 대입한다. 현재 두점(:)은 단어들을 구분하기 위해 사용된다.
4. 변수바꿔넣기를 진행한 다음 for순환은 단어들사이의 내부마당식별자로 두점(:)을 사용하며 매 단어들을 순환한다.
5. 단어목록안의 매 이름들이 현시된다.
6. OLDIFS에 기록된 원래의 값을 IFS에 다시 대입한다.
7. 위치파라메터들을 설정한다. $1에는 Jill, $2에는 Jane 그리고 $3에는 Jolene을 각각 대입한다.
8. $* 는 모든 위치파라메터 Jill, Jone 그리고 Jolene들을 처리한다. for순환은 매 순환을 통해 girl변수에 이름들을 설정한다.

9. 배렬

Korn쉘배렬들은 옹근수나 단어로 이루어 진 1024(크기변화)까지의 원소들을 포함하는 1차원배렬들이다. 첨수는 0으로 시작된다. 배렬의 매 원소는 독자적으로 설정 혹은 해제할수 있다. 값은 특수한 순서로 설정되지 않는다. 실례로 첫번째 원소에 값을 설정하기전에도 10개의 원소에 값을 설정할수 있다. set지령에 –A추가선택을 주어 설정할수 있다. 련상배렬은 1988년이후의 Korn쉘판본들에서 지원된다.

실례 10-118

(The Command Line)

1. $ **array[0]=tom**
   $ **array[1]=dan**
   $ **array[2]=bill**
2. $ print **${array[0]}** *# Curly braces are required.*
   tom
3. $ print **${array[l]}**
   dan
4. $ print ${**array[2]}**
   bill
5. $ print **${array[*]}** *# Display all elements.*
   tom dan bill
6. $ print **${#array[*]}** *# Display the number of elements.*
   3

설명

1. 배렬의 첫 3개 원소에 값을 대입한다. 첨수는 0으로 시작한다.
2. 첫번째 원소의 값 tom이 현시된다. 대괄호{ }안에 변수를 정확히 넣어야 한다. $array[0]은 tom[0]을 현시한다.
3. 두번째 원소의 값 dan이 현시된다.
4. 세번째 원소의 값 bill이 현시된다.
5. 배렬안에 있는 모든 원소값이 현시된다.
6. 배렬안에 있는 원소개수가 현시된다. 만약 사용자가 배렬의 크기와 행을 알고 있으면 배렬은 typeset지령으로 선언할수 있다.

실례 10-119

(At The Command Line)

1. $ **typeset -i ints[4]** *# Declare an array of four integers.*
2. $ ints[0]=50
   $ ints[l]=75
   $ ints[2]=100
3. $ ints[31=happy
   ksh: happy: bad number

설명

1. typeset지령은 4개의 옹근수들을 가진 배렬을 작성한다.

2. 옹근수값들이 배렬에 대입된다.
3. 문자렬값을 배렬의 네번째 원소에 대입하며 쉘은 표준오유에 통보문을 보낸다.

set지령으로 배렬을 창조하기 사용자는 set지령을 사용하여 배렬의 값을 대입할수 있다. -A추가선택다음에 있는 첫번째 단어는 배렬의 이름이며 그 나머지 단어들은 배렬의 원소들이다.

실례 10-120

(The Command Line)

1. $ **set -A fruit apples pears peaches**
2. $ print **${fruit[0]}**
   apples
3. $ print **${fruit[*]}**
   apples pears peaches
4. $ **fruit[1]=plums**
5. $ print **${fruit[*]}**
   apples plums peaches

설명

1. set지령에 –A추가선택을 주면 배렬을 작성할수 있다. 배렬의 이름 fruit는 –A추가선택다음에 있다. fruit배렬의 매개 원소들이 이름뒤에 놓인다.
2. 첨수는 0부터 시작한다. 배렬을 정확히 평가하기 위해서는 변수량열에 대괄호 { }를 주어야 한다. 배렬의 첫번째 원소가 현시된다.
3. 별표(*)는 첨수처럼 사용되며 모든 원소를 현시한다. 별표(*)가 첨수처럼 사용될 때 배렬의 모든 원소들이 현시된다.
4. 배렬의 두번째 원소에 값 plums가 다시 대입된다.
5. 배렬의 모든 원소가 현시된다.

10. 함수

Korn쉘함수는 Bourne쉘에서 쓰는것과 류사하며 프로그람을 모듈화하기 위해 사용한다. 함수는 내부지령과 류사하게 함수이름을 입력하여 류사한 동작을 수행하는 한개이상의 지령집합이다. 여기서는 함수사용에 대한 중요한 몇가지 규칙들을 보여 준다.

1. Korn쉘은 먼저 내부지령, 함수, 그다음 실행가능한것들을 실행한다.
   함수들은 그것들이 정의되여 있을 때에만 기억기안으로 읽어 들이며 항상 그것들을 참고한다.
2. 함수는 사용하기전에 반드시 정의되여 있어야 한다. 그러므로 스크립트를 시작할 때 함수정의들을 배치하는것이 제일 좋다.
3. 함수는 현재환경에서 실행된다. 그것들은 호출된 스크립트와 함께 변수를 공유하며 위치파라메터로 그것들을 설정하여 인수들을 넘겨 준다. 현재작업

등록부라는것은 호출되는 스크립트를 말한다. 만일 함수안에서 등록부를 변경하면 호출스크립트안에서 변경된다.

4. Korn쉘에서 사용자는 typeset지령을 사용하여 함수안에서 국부변수를 선언할수 있다. ksh함수들은 자식쉘들을 반출할수 있다.
5. return명령문은 255를 넘지 않는 함수나 주어진 값안에서 설정된 마지막지령의 탈퇴상태를 되돌려 보낸다.
6. 함수와 정의를 현시하기 위해서는 주어 진 별명 functions를 사용해야 한다.
7. 함수들은 국부변수로 되며 함수를 탈퇴할 때 그 이전값으로 재설정된다 (Bourne쉘에서는 그렇게 되지 않는다.).
8. 함수들은 재귀적으로 호출할수 있다. 즉 자기자체를 호출할수 있다. 재귀호출은 조종된다. Korn쉘은 통보문 <재귀호출이 너무 길다>와 그밖의것들을 경고한다.
9. 함수들은 자동적으로 넣어 질수 있다. 참조하면 함수들이 무조건 정의되고 참조하지 않으면 함수들이 기억기안에 넣어 지지 않는다.
10. 1988년이후에 나온 Korn쉘판본들은 참조에 의해 변수들을 넘기는 함수들과 혼합변수들을 제공한다. 내부지령은 같은 이름의 함수앞에서 인차 찾을수 있다. 그아래 판본들에서는 내부지령을 무시하는 함수를 쓰기 위해 별명과 함수를 조합하여 사용하는것이 필요하다.[8.]

함수정의 함수는 그것이 호출되기전에 정의되여야 한다. Korn쉘함수는 예약어 function과 그뒤에 함수이름을 주어 정의한다. 대괄호{ }들은 매개 괄호의 안쪽에 각각 공백을 주어야 한다(Korn쉘스크립트에 익숙될 때까지 낡은 형의 함수정의를 위해 Bourne쉘함수에서 보여 준다.).

형식

function 함수이름 { 지령들 ; 지령들 }

실례 10-121

```
function usage { print "Usage $0 [-y] [-g] "; exit 1; }
```

설명

함수이름은 usage이다. 진단통보문을 현시하기 위해 사용하는데 스크립트가 인수 –y 혹은 -g를 다 접수하지 않으면 스크립트는 탈퇴한다.

함수현시와 해제 국부함수정의를 현시하기 위해서는 typeset –f를 입력한다. 반출함수정의를 현시하기 위해서는 typeset –fx를 입력하며 함수를 해제하기 위해서는 <unset –f함수이름>을 입력한다. typeset지령을 표 10-30에서 보여 준다.

국부변수와 되돌이값 typeset지령은 국부변수작성에 사용된다. 이 변수들은 그것들이 작성된 함수에서만 쓸수 있다. 함수밖에서는 국부변수가 정의되지 않는다. 함수의 되돌이값은 특수한 rerurn지령을 사용하지 않는 한 스크립트에서 마지막지령의 탈퇴상태값이 된다. 만일 return지령에 값을 설정하면 값은 물음표(?)변수에 기억된다. 그것은 0

8.. 콘 다비드, 볼스키, 모리스 저 《korn shell command and programming language》 (Englewood Cliffs, NJ:Prentice, Inc1988) 77 페지

과 255사이의 옹근수값을 가질수 있다. 수값으로 제한되기때문에 함수의 출력을 귀환시키는데 지령바꿔넣기를 사용할수 있고 UNIX지령의 출력을 얻으려면 변수에 출력을 설정할수 있다.

실례 10-122

(The Script)

```
     # !/bin/ksh
     # Scriptname: do_increment
     # Using the return Command)
1.   function increment {
2.      typeset sum          # sum is a local variable.
           (( sum = $1 + 1 ))
3.         return $sum       # Retrn the value of sum to the script.
     }
     print –n "The sum is "
4.   increment 5             # Call the function increment and pass 5 as a
     # parameter. 5 becomes $1 for the increment
     # function.
5.   print $?                # The return value is stored in the ? variable
6.   print sum               # The variable "sum" was local to the
     # function, and is undefined in the main
     # script. Nothing is printed.
```

(The Output)

```
$ do_increment
5.  The sum is 6
    6
```

설명

1. increment함수가 정의된다.
2. typeset지령은 이 함수에 대한 국부변수 sum을 정의한다.
3. return내부지령에 인수가 있으면 함수가 효력이 있는 행다음에 있는 기본스크립트로 되돌아 가며 물음표(?)변수에 그 인수를 기억한다. 스크립트에서 increment함수는 인수와 함께 호출된다.
4. increment함수는 인수 5로 호출된다.
5. 정확히 인수가 return내부지령에 주어 지지 않으면 함수의 탈퇴상태가 물음표(?)변수에 기억된다. return내부지령인수는 함수를 위한 되돌이상태를 규정하는데 그 값은 물음표(?)변수에 기억된다. 또한 그 값은 0으로부터 255사이의 옹근수여야 한다.
6. sum은 함수 increment에서 국부변수로 정의되였기때문에 함수를 효력이 있게 하는 스크립트에서는 정의되지 않는다. 아무것도 현시하지 않는다.

실례 10-123

(Using Command Substitution)

(The Script)

```
        # Scriptname: do_square
        #!/bin/ksh
1.  function square {
        (( sq = $1 * $1 ))
           print "Number to be squared is $1. "
2.          print "The result is $sq "
        }
3.  read number?"Give me a number to square. "
4.  value_returned=$( square $number)
5.  print $value_returned
```

(The Output)

```
        $ do_square
5.  Number to be squared is 10. The result is 100
```

설명

1. square라는 함수를 정의한다. 인수자체를 두제곱한다.
2. 수의 두제곱결과가 현시된다.
3. 사용자에게 입력을 요구한다.
4. 함수 square는 인수로 수자(사용자로부터 입력된것)와 함께 호출된다. 함수가 화페기호다음에 소괄호 ( )기호안에 들어 있기때문에 지령바꿔넣기된다. 함수출력(2개의 print명령문들)은 변수 value_returned에 대입된다.
5. 지령바꿔넣기는 문자렬 《Number to be squared is》와 《The result is 100》사이에 있는 행바꾸기를 소거한다.

반출된 함수 함수정의는 typeset지령 <typeset –fx 함수이름>으로서 ENV파일에 그것들을 정의하지 않으면 자식쉘에 넘어 가지 않는다. 사용자는 Korn쉘로부터 스크립트까지 또는 한 스크립트에서 다른 스크립트까지 typeset –fx로 함수를 반출할수 있지만 ksh의 한개 호출로부터 다음호출까지는 그렇게 못한다(재촉문에 ksh라고 입력하면 새로운 쉘이 기동된다.). 반출된 함수정의는 새로운 쉘에서 계승되지 않는다.

실례 10-124

(The First Script)

```
      $ cat calling_script
      #!/bin/ksh
1.  function sayit { print "How are ya $l?" ; }
```

```
2.   typeset –fx sayit     # Export sayit to other scripts
3.   sayit Tommy
4.   print "Going to other script"
5.   other_script            # Call other_script
     print "Back in calling script"
       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
(The Second Script)
     $ cat other_script
                  NOTE: This script cannot be invoked with #!/bin/ksh
6.   print "In other script "
7.   sayit Dan
8.   print "Returning to calling script"
(The Output)
     $ calling_script
3.   How are ya Tommy?
4.   Going to other script
6.   In other script
7.   How are ya Dan?
8.   Returning to calling script
     Back in calling script
```

설명

1. 함수 sayit가 정의된다. 그것은 $1에 기억시키기 위한 한개 인수를 설정한다.
2. typeset지령에 –fx추가선택을 주면 이 스크립트로부터 호출되는 임의의 스크립트가 반출되도록 함수를 허락한다.
3. 함수 sayit는 인수로 Tommy와 함께 인용된다. Tommy는 함수에서 $1에 기억된다.
4. sayit함수가 끝난 다음 프로그람은 여기서부터 다시 시작한다.
5. other_script로 불리우는 스크립트가 실행된다
6. 현재 다른 스크립트안에 있다. 이 스크립트는 첫번째 스크립트 sayit로부터 호출된다. 이 행은 ksh자식쉘을 기동시키기때문에 행 #!/bin/ksh와 함께 기동할수 없으며 Korn쉘이 호출되면 함수반출을 하지 못한다.
7. 함수 sayit가 인용된다. Dan이 함수에서 $1에 기억되는 인수로 넘겨 진다.
8. 이 행을 현시한 다음 other_script가 끝나고 조종은 함수가 호출된 다음의 왼쪽에 있는 행에서 스크립트를 호출하기 위해 되돌아 간다. 조종이 실행된다.

11. typeset지령과 함수추가선택

typeset지령은 함수속성을 현시하기 위해 사용된다.

표 10-30. typeset지령과 함수추가선택

추가선택	무엇을 하는가
typeset –f	모든 함수들과 그 값들을 현시한다. 모든 함수정의들을 기억한 리력파일을 가져야 한다.
typeset +f	함수이름들을 정확히 현시한다.
typeset –fx	쉘스크립트들이 교차로 반출하는 모든 함수정의들을 현시한다.
typeset –fu func	func는 아직 정의되지 않은 함수이름이다.

자동넣기된 함수 자동넣기된 함수는 그것을 참조할 때까지 프로그람안에 넣어 지지 않는다. 자동넣기되는 함수는 파일의 그 어디에서라도 정의될수 있으며 작으면서 압축된 스크립트들을 보호하는데서는 나타나지 않는다. 자동넣기를 사용하기 위해서는 ENV파일안에 FPATH방식을 설정하는것이 필요하다. FPATH변수는 함수파일들을 포함하는 등록부들을 위한 탐색경로를 포함하고 있다. 이 등록부에서 파일들은 그안에서 정의된 함수로 같은 이름을 가진다. typeset –fu에 대한 autoload별명들은 아직 자동넣기되는 함수로 정의되지 않은 함수이름들을 규정한다. autoload지령의 인수로 함수를 가지고 실행된 다음 그안에 포함되여 있는 지령들을 실행하기 위해 함수를 효력이 있게 한다. 자동넣기함수들의 원래의 우점은 더 좋게 실행하자는것이기때문에 그것이 참조되지 않는 한 Korn쉘은 함수정의를 읽지 못한다.[9]

실례 10-125

(The Command Line)
1. $ **mkdir functionlibrary**
2. $ cd functionlibrary
3. $ **vi foobar**

(In Editor)
4. **function foobar { pwd; ls; whoami; }** *# function has the same*
   # name as the file.

(In .profile File)
5. **export FPATH=$HOME/functionlibrary** *# This path is searched for*
   # functions.

(In Your Script)
6. **autoload foobar**
7. **foobar**

설명

1. 함수를 기억하는 등록부를 작성한다.
2. 등록부에로 이행한다.
3. foobar는 functionlibrary안에 있는 파일이다. foobar파일은 함수 foobar의 정의를 포함한다. 파일이름과 함수이름은 서로 대응된다.

---

9. 볼스키, 모리스, 콘다비드 저 《The New Kornshell.》(Upper Sa--ddle River, NJ:Prentice Hall, 1995), 78페지

4. 함수 foobar는 foobar라는 파일에서 정의된다.
5. 사용자의 .profile초기화파일에서 FPATH변수에는 함수가 기억된 경로를 대입한다. 이것은 함수가 자동넣기될 때 Korn쉘이 검색하는 경로이다. FPATH를 반출한다.
6. 스크립트에서 함수 foobar가 프로그람의 기억기안으로 들어 온다.
7. 함수 foobar를 실행한다.

많은 함수들이 한개 파일에 기억된다. 실례로 계산함수들은 math라는 파일에 들어 있다. 함수는 그것을 기억한 똑 같은 이름을 가져야 하기때문에 함수파일을 고정련결해야 한다. 이름은 함수를 정의한 파일에로 련결된다. 실례로 math파일에서 함수가 square라고 불리 우면 square의 다른 이름을 math파일에 주기 위해서는 UNIX ln명령을 사용해야 한다. math파일과 square파일은 참조할수 있으며 둘다 대응하는 함수이름으로 파일을 참고할수 있다. square함수는 그자체의 이름으로 자동넣기될수 있다.

실례 10-126

(The Command Line)

1. $ In math squareadd divide
2. $ ls –i
   12256 add
   12256 math
   12256 square
   12256 divide
3. $ autoload square; square

설명

1. UNIX ln(link)지령은 바꿔넣기하려는 이름을 파일에 준다. math파일과 square는 같은 파일이다. 련결계수값은 매 련결을 참고할 때마다 하나씩 증가된다.
2. 목록은 그것들이 모두 한개 파일이지만 여러가지 이름으로 호출될수 있다는것을 의미하는 같은 식별자번호를 가지는 모든 파일들을 보여 준다.
3. 이제 square파일이 자동넣기될 때 함수 square는 같은 이름을 가지고 실행될것이다. 파일에서 다른 이름으로 정의하지 않는 한 이름에 의해 특수하게 자동넣기될 때까지 참조될수 있다.

12. 신호트랩프

프로그람이 실행될 때 Control-C 혹은 Control-\ 이 눌리우면 프로그람은 곧 신호를 붙여 끝나게 된다. 신호가 발생한 다음 직접 프로그람을 끝내지 못하므로 일정한 시간이 걸린다. 신호를 무시하고 프로그람을 실행하거나 스크립트를 자동적으로 탈퇴하기 전에 조작을 소거하기 위한 몇가지 분류를 실행한다. trap지령은 신호를 접수할 때 프로그람의 특성을 조종할수 있게 한다. 신호는 한 프로쎄스로부터 다른 프로쎄스로 보낼수 있는 번호로 이루어 진 비동기통보문으로 정의되거나 중요한 건들이 눌리거나 일부 우연

현상이 일어 나는 프로쎄스에 따라 조작체계에 의해 정의된다.[10.] trap지령은 신호를 접수한 기초우에서 실행되는 현재지령을 끝내기 위해 쉘을 호출한다. trap지령다음에 신호내부에 인용부호안의 지령들이 있으면 그 지령들은 특수한 신호를 접수한 기초우에 식별된다. 모든 신호들의 목록과 그것에 응답하는 번호를 얻기 위해서는 kill –l지령을 사용해야 한다.

형식

trap‘ 지령 ; 지령’signal

실례 10-127

trap 'rm tmp*$$; exit 1’ 1 2 15

설명

신호1(정지), 2(새치기) 혹은 15(소프트웨어끝내기)가운데서 임의의것이 설정되는 모든 tmp파일을 소거한 다음 탈퇴한다.

만일 스크립트가 실행될 때 새치기가 들어 오면 trap지령은 여러가지 방법으로 새치기신호를 조종한다. 사용자는 기정값이나 혹은 신호의 무시 또는 신호를 줄 때 호출되는 조종함수를 작성한다. 신호번호들과 대응하는 이름들에 관한 목록을 표 10-31에서 보여준다 .

표 10-31. 신호[ㄱ](형태: kill –l)

1) HUP	12) SYS	23) POLL
2) INT	13) PIPE	24) XCPU
3) QUIT	14) ALRM	25) XFSZ
4) ILL	15) TERM	26) VTALRM
5) TRAP	16) URG	27) PROF
6) IOT	17) STOP	28) WINCH
7) EMT	18) TSTP	29) LOST
8) FPE	19) CONT	30) USR1
9) KILL	20) CHLD	31) USR2
10) BUS	21) TTIN	
11) SEGV	22) TTOU	

ㄱ. 이 지령의 출력은 조작체계에 따라 조금씩 차이난다.

거짓신호 3개의 거짓신호는 실지신호는 아니지만 프로그람을 오유수정하는데 방조를 주기 위해 쉘에 의해 발생한다. 이것들은 trap지령에 의해 실지 신호처럼 취급되며 같은 방법으로 정의된다. 거짓신호들에 대한 목록을 표 10-32에서 보여 준다.

10.. 콘 다비드, 볼스키, 모리스 저 《korn shell command and programming language》(Englewood Cliffs, NJ:Prentice, Inc1988) 327 페지

표 10-32. Korn쉘거짓트랩프신호

신 호	무엇을 하는가
DEBUG	모든 스크립트지령다음에 trap지령이 실행된다.
ERR	스크립트안에 있는 임의의 지령이 0이 아닌 탈퇴 상태값을 되돌리면 trap지령이 실행된다.
0 혹은 EXIT	쉘이 탈퇴하면 trap지령이 실행된다.

HUP와 INT와 같은 신호이름은 보통 SIG가 앞에 붙는다. 실례로 SIGHUP, SIGINT 등으로 된다. Korn쉘은 보통 SIG앞붙이가 없이 신호이름을 주었으나 수값을 사용할수 있는 신호에 대하여서는 기호이름을 사용하도록 한다. 실례 10-128에서 보여 준다.

신호의 재설정 기정특성으로 신호를 재설정하기 위해서는 trap지령다음에 신호이름 혹은 번호를 주어야 한다. 함수안에서 설정된 trap지령들은 함수를 국부함수로 취급한다. 즉 그것들은 그것이 설정된 함수밖에서는 쓸수 없다.

실례 10-128

trap 2 *or* **trap INT**

설명

신호2 즉 SIGINT를 위한 기정동작을 재설정한다. 기정동작은 새치기건(Control-C)을 누르면 처리를 중단시킨다.

신호의 무효 trap지령다음에 null 인용부호(“ ”)가 있으면 현시된 신호는 프로쎄스에 의해 무시된다.

실례 10-129

trap “ “1 2 *or* **trap "" HUP INT**

설명

신호1(SIGHUP)과 신호2(SIGINT)는 쉘프로쎄스에 의해 무시된다.

실례 10-130

```
(The Script)
    #i/bin/ksh
    # Scriptname: trapping
    # Script to illustrate the trap command and signals
    # Can use the signal numbers or ksh abbreviations seen
    # below. Cannot use SIGINT, SIGQUIT, etc.
1.  trap ‘print "Control-C will not terminate $PROGRAM."' INT
2.  trap ‘print "Control-\  will not terminate $PROGRAM."' QUIT
3.  trap ‘print "Control-z will not terminate $PROGRAM."' TSTP
```

```
4.   print "Enter any string after the prompt.\
      When you are ready to exit, type \ "stop\ "."
5.   while true
      do
6.          print -n "Go ahead.. > "
7.          read
8.          if [[ $REPLY = [Ss]top ]]
          then
9.               break
           fi
10.   done
```

(The Output)
$ **trapping**

4. *Enter any string after the prompt.*
   When you are ready to *exit, type "stop".*
6. *Go ahead..,> this is it^**C***
1. ***Control-C will not terminate trapping.***
6. *Go ahead...> this is it again**^Z***
3. **Coatrol-Z will not terminate trapping.**
6. *Go ahead...> this is never it^*
2. ***Control-\   will not terminate trapping.***
6. *Go ahead... > **stop***
   $

설명

1. 첫번째 trap는 INT신호 즉 Control-C에 해당된다. 만약 프로그람이 실행되는 도중에 Control-C를 누르면 인용부호안에 있는 지령이 실행된다. 중지할 대신에 프로그람은《 Control-C will not terminate trapping》을 현시하고 재촉문에서 사용자입력을 계속 진행한다.
2. 두번째 trap지령은 사용자가 Cotrol-\ 즉 QUIT신호를 처리할 때 실행된다. 문자렬 《Cotrol-\ will not terminate trapping》이 현시되고 프로그람은 실행을 계속한다. 이 신호 즉 SIGQUIT는 기정으로 프로쎄스를 중지하고 null파일을 작성한다.
3. 세번째 trap지령은 사용자가 Control-Z 즉 TSTP신호를 처리할 때 실행된다. 문자렬 《Control-Z\ will not terminate trapping》이 현시되고 프로그람은 실행을 계속한다. 이 신호는 보통 일감조종이 실행되면 배경안에 프로그람이 생기게 한다.
4. 사용자에게 입력을 재촉한다.
5. 첫번째 trap는 INT신호 즉 Control-C에 해당된다. 만약 프로그람이 실행되는 도중에 Control-C를 누르면 인용부호안에 있는 지령이 실행된다. 중지할 대신에 프로그람은《 Control-C will not terminate trapping》을 현시하고 재촉문에서 사용자입력을 계속 진행한다.

6. 두번째 trap지령은 사용자가 Cotrol-\ 즉 QUIT신호를 처리할 때 실행된다. 문자렬 《Cotrol-\ will not terminate trapping》이 현시되고 프로그람은 실행을 계속한다. 이 신호 즉 SIGQUIT는 기정으로 프로쎄스를 중지하고 null파일을 작성한다.
7. 세번째 trap지령은 사용자가 Control-Z 즉 TSTP신호를 처리할 때 실행된다. 문자렬 《Control-Z\ will not terminate trapping》이 현시되고 프로그람은 실행을 계속한다. 이 신호는 보통 일감조종이 실행되면 배경안에 프로그람이 생기게 한다.
8. 입력을 위해 사용자가 재촉문에 표시된다.
9. while순환에 들어 간다.
10. 문자렬 Go ahead >가 현시되고 프로그람은 입력을 대기한다(read다음 행에서 보여 준다.).
11. read지령은 내부변수 REPLY에 사용자입력을 대입한다.
12. REPLY의 값이 Stop 혹은 stop이면 break지령은 순환을 탈퇴하게 하며 프로그람은 끝나게 된다. Stop이나 stop의 입력은 kill지령에 의하여 중지하지 않으면 프로그람을 끝나게 하는 한가지 방법이다.
13. break지령은 순환본체를 탈퇴하도록 한다.
14. done예약어는 순환의 끝을 알려 준다.

실례 10-131

```
(The Script)
$ cat trap.err
#!/bin/ksh
# This trap checks for any command that exits with a non-zero
     # status and then prints the message.
1.  trap 'print "You gave me a non-integer. Try again. "' ERR
2.  typeset –I number        #Assignment to number must be integer
3.  while true
    do
4.  print –n "Enter an integer. "
5.  read –r number 2> /dev/null
6.       if (($?== 0 ))                 # Was an integer read in?
         then                           # Was the exit status zero?
7.  break
         fi
    done

8.  trap – ERR                  # Unset pseudo trap for ERR
     n=$number
9.  if grep ZOMBIE /etc/passwd > /dev/null 2>&1
    then
         :
    else
10.  print "\ $n is $n. So long"
```

```
fi
(The Output)
    $ trap.err
4.  Enter an integer, hello
1.  You gave me a non-integer. Try again.

4.  Enter an integer. Good-bye
1.  You gave me a non-integer. Try asrain .
4.  Enter an Integer. \ \ \
1.  You gave me a non-integer. Try again .
4.  Enter an Integer.5
10. $n is 5. So long.

$ trap .err
4.  Enter an Integer. 4.5
10. $n is   4. So long.
```

설명

1. ERR(거짓 혹은 위조)신호는 탈퇴상태를 0이 아니게 되돌리는 프로그람에서 2중인용부호안에 있는 통보문을 인쇄한다. 즉 실패이다.
2. typeset지령에 –i추가선택을 주면 다만 옹근수만을 대입할수 있는 옹근수변수 즉 number를 참조한다.
3. true지령의 탈퇴상태는 항상 0이며 while순환본체에 들어 간다.
4. 사용자에게 옹근수로 입력하겠는가를 문의한다.
5. read지령은 사용자입력을 읽고 number변수에 그것을 대입한다. 수값은 옹근수여야 하며 그렇지 않으면 오유통보문이 /dev/null에 전송된다. read지령에 –r추가선택을 주면 –값을 넣게 한다(미누스 즉 -부호로 시작하는).
6. read지령의 탈퇴상태가 0이면 수값이 넣어 지며 if명령문이 실행된다.
7. break지령이 실행되고 순환을 탈퇴한다.
8. 거짓 korn쉘신호 ERR를 위한 trap가 해제된다.
9. grep가 실패할 때 0이 아닌 탈퇴상태를 되돌린다. 만약 ERR트랩프를 해제하지 않은 상태에서 grep가 /etc/passwd파일에서 ZOMBIE찾기를 실패하면 스크립트는 《You gave me a non-integer. Try again.》을 현시한다.
10. 이 행은 grep가 실패하면 현시된다. 류동소수점 4.5를 입력하면 수값은 옹근수를 취한다.

트랩프와 함수 함수에서 trap를 사용하면 trap와 그 지령들은 함수에서 국부적으로만 쓰인다.

실례 10-132

```
(The Script)
    #!/bin/ksh
1.  function trapper {
        print "In trapper"
2.      trap 'print "Caught in a trap!"' INT
```

```
            print "Got here."
            sleep 25
        }
3.   while :
     do
          print "In the main script"
4.         trapper     # Call the function
5.         print "Still in main"
          sleep 5
          print "Bye"
     done
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(The Output)
$ functrap
In the main script
In trapper
Got here.
^CCaught in a trap!
$
```

설명

1. 함수 trapper가 정의된다. trapper지령을 포함한다.
2. Control-C 를 누르면 trap지령이 실행된다. trap지령안에서 print지령이 실행되며 프로그람은 실행을 계속한다. sleep지령이 실행되는 도중에 Control -C가 입력된다. 보통 프로그람은 새치기지령다음부터 즉시에 실행을 계속한다(sleep지령은 제외하고). trap지령은 4행다음부터는 효력을 가지지 않는다.
3. 스크립트의 기본부분에서 while순환이 시작된다. 두점(：)은 항상 탈퇴상태에 0을 되돌리는 아무런 동작도 하지 않는 지령이다. 순환은 무한순환으로 된다.
4. 순환에서 함수 trapper를 호출한다.
5. trapper함수안에서 trap지령은 함수에서만 국부적으로 사용되기때문에 이 프로그람부분에서는 효력을 가지지 않는다. 함수가 표준방법(^C는 처리되지 않는다.)으로 탈퇴하면 여기서부터 실행을 계속한다. ^C의 기정적인 특성은 신호가 여기 혹은 아래에 있는 어떤 행들에 보내지면 스크립트를 중지시킨다.

13. 협동프로세스

협동프로세스는 다른 지령의 표준입력에 쓰거나 표준출력으로부터 읽도록 쉘스크립트를 허용하는 특수한 두가지 방법의 파이프행이다. 이것은 존재하는 프로그람의 새로운

대면부를 작성하는 방법을 제공한다. 추가연산자 즉 \ &은 협동프로쎄스로서 지령을 초기화하기 위해 지령의 끝에 놓여야 한다. 보통 방향바꾸기와 배경프로쎄스는 협동프로쎄스를 사용하지 않는다. print와 read지령은 협동프로쎄스로부터 읽거나 쓰기 위해 –p추가선택을 요구한다. 출력은 표준출력에 보내지며 출력의 매 통보문의 끝에는 행바꾸기가 있어야 한다. 표준출력에로 보낸 매 통보문다음에 표준출력을 놓아야 한다. 사용자는 exec지령에 >&p 혹은 <&p연산자를 사용하여 다중협동프로쎄스를 실행시킬수 있다. 협동프로쎄스로 파일서술자 4를 열기 위해서는 exec 4>&p를 입력하여야 한다.

실례 10-133

(The Script)

```
    # !/bin/ksh
    # Scriptname: mycalculator
    # A simple calculator -- uses the bc command to perform the
    # calculations
    # this program allows
    # you to uses floating point numbers by writing to and reading
    # from the bcprogram.

1.  cat << EOF
    ***************************************************************
2.       WELCOME TO THE CALCULATOR PROGRAM
    ***************************************************************
3.  EOF
4.  bc |&                   # Open coprocess
5.  while true
    do
6.       print "Select the letter for one of the operators below"
7.       cat << - EOF
             a) +
             s) –
             m) *
             d) /
             e) ^
         EOF
8.       read op
9.       case $op in
           a) op="+";;
           s) op="-";;
           m) op="*";;
           d) op="/";;
           e) op="^"
           *) print "Bad operator"
                 continue;;
```

```
        esac
10.     print –p scale=3                      # write to the coprocess
11.     print "Please enter two numbers:      # write to standard out
12.     read num1 num2                        # read from standard in
13.     print –p "$num1" "$op" "$num2"        # write to the coprocess
14.     read –p result                        # read from coprocess
15.     print $result
16.     print –n "Continue (y/n)? "
17.     read answer
18.     case $answer in
        [Nn]* )
19.         break
20.     done
21.     print Good-bye
```

(The Output)

$ **mycalculator**

```
   *****************************************************************
1.          WELCOME TO THE CALCULATOR PROGRAM
   *****************************************************************
6.     Select one of the operators below
7.              a) +
                s) –
                m) *
                d) /
                e) ^
   e
11   Please enter two numbers :
     2.3 4
     27.984
16   Continue (y/n)? y
6    Select one of the operators below
7.          a) +
            s) –
            m) *
            d) /
            e) ^
   d
11   Please enter two numbers:
     2.1 4.6
     0.456
16   Continue (y/n)? y
6    select one of the operators below
```

```
7        a) +
         s) -
         m) *
         d) /
         e) ^
   m
11  Please enter two numbers:
     4 5
    20
16  Continue (y/n)? n
    Good-bye
```

설명

1. here문서가 차림표를 현시하기 위해 사용된다.
2. 이 본문은 차림표아래의 머리부로 현시된다.
3. EOF는 here문서의 끝을 알려 주는 사용자정의끝지적자이다.
4. bc지령(탁상수산기)이 협동프로쎄스로 열려 진다. 배경에서 실행된다.
5. while순환이 시작된다. true지령은 항상 0이라는 성공적인 탈퇴상태를 되돌리기때문에 순환은 break 혹은 exit가 나타날 때까지 계속 진행한다.
6. 현시된 차림표로부터 항목을 설정하기 위해 재촉문이 현시된다.
7. 다른 here문서는 사용자가 bc프로그람을 위해 선택하는 산수연산목록을 현시한다.
8. read지령은 변수 op에 사용자입력을 대입한다.
9. case지령에는 op변수가운데서 한개 값이 대응되며 op에 연산수를 대입한다.
10. print지령에 –p추가선택, 파이프출력 즉 scale=3을 설정하였다. bc지령은 print출력을 입력으로 접수하고 scale에 3을 설정한다(scale은 bc에 의해 현시되는 수의 소수부를 10진수로 정의한다.).
11. 2개의 수값을 입력하기 위해 재촉문이 나타난다.
12. read지령은 사용자입력을 변수 num1 그리고 num2에 대입한다.
13. print-p지령은 산수연산표현식을 bc협동프로쎄스에 보낸다.
14. 쉘은 bc협동프로쎄스(read –p)로부터 읽고 입력을 변수 result에 대입한다.
15. 계산결과($result)가 현시된다.
16. 사용자에게 계속 하겠는가를 묻는다.
17. 사용자가 입력한다. 그것을 변수 answer에 대입한다.
18. case명령문은 변수 answer를 평가한다.
19. 사용자가 No 혹은 no 혹은 nope로 입력하면 break지령이 실행되며 while순환은 완료되고 조종은 21행으로 넘어 간다.
20. done예약어는 while순환의 끝을 알려 준다.
21. 이 행은 순환이 완료될 때 현시된다.

14. 오유수정

ksh지령에 noexec추가선택을 설정하거나 –n인수를 사용하면 실지로 지령을 실행시킴이 없이 스크립트물음표를 검사할수 있다.

스크립트에 문법오유가 있으면 쉘은 오유를 알려 주며 오유가 없으면 아무것도 현시하지 않는다.

일반적으로는 xtrace추가선택을 설정하거나 ksh지령에 –x추가선택을 주어 스크립트의 오유수정을 검사하는 방법을 사용한다. 이 추가선택들은 스크립트의 추적을 허용한다.

스크립트로부터 매 지령의 변수바꿔넣기가 진행된 다음 지령이 실행된다. 스크립트로부터 한개 행이 현시될 때 PS4재촉문 즉 더하기(+)기호이다.

PS4재촉문을 변경시킬수 있다. verbos추가선택을 설정하거나 korn쉘에 –v추가선택(ksh –v스크립트가름)을 주면 스크립트의 매행이 현시되고 실행된다. 오유수정지령은 표 10-33에서 보여 준다.

표 10-33. 오유수정지령과 추가선택

지 령	기 능	어떻게 동작하는가
ksh -x 스크립트가름	echo추가선택과 함께 ksh를 호출한다.	실행하기전에 변수바꿔넣기를 한 다음 스크립트의 매행을 현시한다.
ksh -v 스크립트가름	verbose추가선택과 함께 ksh를 호출한다.	지령을 입력한 즉시 집행하기전에 스크립트의 매행을 현시한다.
ksh -n 스크립트가름	noexec추가선택과 함께 ksh를 호출한다.	해석하지만 지령을 실행하지 않는다.
set -x 혹은 set -o xtrace	echo추가선택을 실행한다.	스크립트에서 실행을 추적한다.
set +x	echo추가선택을 해제한다.	추적을 해제한다.
typset –ft	추적을 설정한다.	함수에서 실행을 추적한다.
export PS4='$LINENO'	기정으로 PS4재촉문이 기호 +이다.	사용자가 재촉문을 재설정할수 있다. 실례로 행번호가 매행에 현시된다.
trap "print $LINENO" DEBUG	스크립트에서 매행의 $LINENO값을 현시한다.	매 스크립트지령을 위해 추적동작이 집행된다. trap지령의 시작에서 보여 준다.
trap 'print Bad input' ERR		령이 아닌 탈퇴상태가 되돌아 오면 trap지령이 실행된다.
trap 'print Exiting from $o' EXIT		스크립트나 함수가 탈퇴할 때 통보문을 현시한다.

실례 10-134

(The Script)

```
    #! /bin/ksh
    # Scriptname: todebug
1.  name="Joe Blow"
2.  if [[ $name = [Jj]* ]] then
```

```
            print Hi Sname
            fi
         num=l
3.   while (( num < 5 ))
        do
4.          (( num=num+l ) )
        done
5.   print The grand total is $num
```

(The Output)

```
1.   $ ksh -x todebug
2.   + name=Joe Blow
     + [{ Joe Blow = [Jj]* ]]
     + print Hi Joe Blow
     Hi Joe Blow
     + num=1                          The + is the PS4 prompt
     + let num < 5
     + let num=num+l
     + let num < 5
     + let num=num+l
     + let num < 5
     + let num=num+l
     + let nmn < 5
     + let nufli=num+l
     + let num < 5
     + print The grand total is 5
      The grand total is 5
```

설명

1. korn쉘은 –x추가선택을 사용한다. 현시방식을 설정한다. 현시장치에 스크립트의 매행의 실행결과가 현시된다. 변수바꿔넣기가 실행된다. –x추가선택은 지령행 즉 #!/bin/ksh –x대신에 스크립트에서 사용될수 있다.
2. 행앞에는 PS4재촉문 즉 더하기(+)기호가 붙는다.
3. while순환에 들어 간다. 4번 순환한다.
4. num값이 하나씩 증가한다.
5. while순환에서 탈퇴한 다음 이 행이 현시된다.

실례 10-135

```
(The Script)
#!/bin/ksh
# Scriptname: todebug2
```

1. trap 'print "num=$num on line $LINENO"' DEBUS

```
num=l
while ( ( num < 5 ) )
   do
   ( ( nuin=num+l ) )
done
print The grand total is $num
```

(The Output)

```
   $ todebug2
```

2. *num=l on line 3*

num=l on -line 4

num=2 on line 6

num=2 on line 4

num=3 on line 6

num=3 on line 4

num=4 on line 6

num=4 on line 4

num=5 on line 6

num=5 on line 4

The grand total is 5

num=5 on line 8

num=5 on line 8

설명

1. LINENO는 현재스크립트행의 번호를 가지고 있는 특수한 korn쉘변수이다. DEBUG신호를 trap지령과 함께 사용하면 실행되는 스크립트에서 매번 지령들을 실행하기 위해 단일인용부호안에 있는 문자렬을 발생시킨다.
2. while순환이 실행됨으로써 변수num의 값과 스크립트행이 현시된다.

15. getopts에서 지령행추가선택처리

만일 사용자가 지령행추가선택번호를 요구하는 스크립트를 작성한다면 위치파라메터가 항상 가장 효과적인 방법으로는 되지 않는다. 실례로 UNIX ls지령은 지령행추가선택번호와 인수를 가지고 있다(추가선택은 횡선 즉 -를 요구하고 인수는 요구하지 않는다.). 추가선택은 여러가지 방법 즉 ls –laFi, ls –i –a –l –F, ls –ia –F 등으로 프로그람에 넘길수 있다. 사용자가 인수와 인수들을 따로따로 처리하게 하는 위치파라메터들을 요구하는 스크립트를 가지고 있으면 ls –l -i –f와 같다. 매 횡선(-)추가선택은 $1, $2 그리고 $3에 기억되지만 사용자가 한개 횡선(-)추가선택 즉 ls –liF?로 모든 추가선택들을 현시하면 –liF는 스크립트에서 $1에 모든것을 대입한다. getopts함수는 ls프로그람에 의하여 처

리된것과 같은 방법으로 지령행추가선택들과 인수들을 처리하도록 한다.[11.] getopts함수는 다양한 조합을 사용하여 인수를 처리하는 runit프로그람을 허용한다.

실례 10-136

(The Command Line)

1. **runit -x -a 200 filex**
2. $ **runit -xn200 filex**
3. $ **runit –xy**
4. $ **runit -yx -n 30**
5. $ **runit -n250 -xy filey**
6. (any other combination of these arguments)

설명

1. runit프로그람은 4개의 인수를 가진다. 즉 x는 추가선택이고 n은 수값인수를 요구하는 추가선택이고 filex는 단독으로 설정하는 인수이다.
2. runit프로그람은 추가선택 x 그리고 n, 수값인수 200으로 조합된다. filex도 인수이다.
3. runit프로그람은 x와 y추가선택을 조합한다.
4. n추가선택은 수값인수 30으로 분리되여 넘겨 진다.
5. runit프로그람은 수값인수와 n추가선택을 조합하고 x와 y도 조합하며 filey는 분리된다.

runit프로그람의 모든 항목값을 얻기전에 getopts가 어떻게 인수들을 처리하는가를 보는데 사용하는 프로그람으로부터 행을 시험해 보아야 한다. 다음의것은 runit라는 스크립트로부터의 행이다.

while getopts :xyn: name

1. x, y 그리고 n은 추가선택이다.
2. 추가선택들은 –아니면 + 둘중의 하나로 시작하는 지령행에서 입력된다.
3. + 혹은 –를 포함하지 않는 임의의 추가선택들은 추가선택목록이 끝이라는것을 getopts에 알려 준다.
4. 추가선택다음에 있는 두점(:)은 추가선택이 인수를 요구한다는것을 의미한다. 즉 –n추가선택은 인수를 요구한다.
5. 추가선택목록앞에 있는 두점(:)은 적합하지 않는 추가선택을 입력할 때 그것을 알려 준다. getopts는 이것을 조종하기 위한 프로그람을 허용한다. 실례로 runit –p지령에서 –p는 적합한 추가선택이 아니며 getopts는 그것을 프로그람적으로 알려 준다. 쉘은 오유통보문을 현시하지 않는다.
6. 매번 getopts를 호출하며 변수name에서 횡선(-)없이 그것을 찾기 위해 다음추가선택을 배치한다(사용자는 임의의 변수이름을 여기에서 사용할수 있다.). 추가선택앞에 더하기(+)기호가 있으면 더하기(+)기호와 함께 name으로 간다. 적합하지 않는 추가선택이 주어 지면 name에 물음표(?)가 대입된

11.. C서고함수 getopt에 대해서는 UNIX 안내서 3장에서 보여 주었다.

다. 인수가 빠졌으면 name에는 두점(:)이 대입된다.

7. OPTIND는 하나로 초기화되는 특수한 변수이며 getopts가 처리하는 다음 인수의 번호로 매번 지령행인수처리를 끝낼 때마다 하나씩 증가한다.
8. OPTARG변수는 적합한 인수값을 포함하며 적합한 추가선택이 주어 지면 적합하지 않은 추가선택값을 OPTARG에 기억한다.

getopts스크립트실례 다음의 실례는 getopts가 어떻게 인수를 처리하는가를 보여 준다.

실례 10-137

```
(The Script)
#!/bin/ksh
# Program optsl
 # Using getopts — First try —
1.  while getopts xy options
      do
2.  case $options in
3.        x) print "you entered -x as an option"
           y) print "you entered -y as an option" esac done
   ----------------------------------------
   (The Command Line)
4.  $ optsl -x
     you entered -x as an option
5.  $ optsl -xy
     you entered -x as an option
     you entered, -y as an option
6.  $ optsl -y
     you entered -y as an option
7.  $ optsl -b
          optsl[3]: getopts: b bad option(s)
8.  $ optsl b
```

설명

1. getopts지령은 while지령의 조건으로 사용된다. 이 프로그람을 위한 유효한 추가선택들을 getopts지령다음에 보여 준다. 그것들은 x와 y이다. 매 추가선택은 순환본체에서 검사된다. 매 추가선택은 횡선(-)없이 변수 option에 대입된다. 그 어떤 인수도 주어 지지 않으면 getopts는 0아닌 상태로 탈퇴하고 while순환은 완료된다.
2. case지령은 x 혹은 y 둘중의 하나를 추가선택변수에서 찾기 위해 매개 가능한 추가선택들을 검사한다.
3. x가 추가선택이면 문자렬 you entered x as an option이 현시된다.
4. 지령행에서 opts1스크립트에 getopts에 의하여 처리되는 적합한 x추가선택이 주어 진다.

5. 지령행에서 opts1스크립트에 getops에 의하여 처리되는 적합한 xy추가선택이 주어 진다.
6. 지령행에서 opts1스크립트에 getopts에 의하여 처리되는 적합한 y추가선택이 주어 진다.
7. opts1스크립트에 적합하지 않은 추가선택인 b추가선택이 주어 진다. getopts는 오유통보문을 보낸다.
8. 덜기(-) 혹은 더하기(+)가 없는 추가선택은 추가선택이 아니라는것을 의미하며 인수처리를 중지하도록 getopts에 알려 준다.

실례 10-138

(The Script)

```
#!/bin/ksh
# Program opts2
# Using getopts — Second try --
1.   while getopts :xy options
      do
2.          case $options in
              x) print "you entered -x as an option";;
              y) print "you entered -y as an option";;
3.          \ ?) print $OPTARG is not a valid option 1>&2;;
              esac
            done
```

(The Command Line)

```
$ opts2 -x
you entered -x as an option

$ opts2 -y
you entered -y as an option
$ opts2 xy
$ opts2 -xy
     ou entered -x as an option
     ou entered -y as an option

4.  $ opts2 -g
     g is not a valid option

5.  $ opfs2 -c
     c is not a valid option
```

설명

1. 추가선택목록앞에 있는 두점(:)은 korn쉘이 틀린 추가선택에 대한 오유통보문을 현시하지 않도록 한다. 그러므로 추가선택이 틀린 추가선택이면 물음표 (?)가 options변수에 대입된다.
2. 표준오유에 자체오유통보문을 현시하도록 허용하는 물음표(?)를 검사하는데

case지령을 사용할수 있다.
3. options변수에 물음표가 대입되면 case명령문이 실행된다. 물음표(?)는 거꿀빗선(\ )에 의하여 보호되기때문에 korn쉘은 통용기호로서 그것을 볼수 없으며 파일이름바꿔넣기를 실행한다.
4. g는 적합한 추가선택이 아니다. 물음표(?)가 options변수에 대입되고 OPTARG에 적합하지 않은 추가선택 g를 대입한다.
5. c는 적합한 추가선택이 아니다. 물음표(?)는 options변수에 대입되며 OPTARG에는 적합하지 않은 추가선택 c를 대입한다.

실례 10-139

```
(The Script)
#I/bin/ksh
# Program opts3
# Using getopts — Third try —
1.  while getopts id options
     do
         case $options in
2.       d) print -R "~d is the ON switch";;
3.       +d) print -R "+d is the OFF switch";;
         \ ?) print $OPTARG is not a valid option;;
           esac
           done
     # Need the —R option with print ox the 쉘 tries to use -3 as a.
     # print option
-------------------------------------
 (The Command Line)
4.  $ opts3 -d
     -d Is the ON switch
5.  $ opts3 +d
     +d is the OFF switch
6.  $ opts3 -e
     e is not a valid option
7.  $ opts3 e
```

설명

1. while지령은 getopts의 탈퇴상태를 검사하며 getopts가 인수를 성공적으로 처리할수 있다면 탈퇴상태 0으로 되돌리고 while순환본체에 입력된다. 만일 사용자가 getopts에게 적합하지 않는 추가선택을 입력하면 오유통보문을 현시하지 않도록 추가선택앞에 있는 두점(:)이 알려 준다.
2. 적합한 추가선택들중의 하나가 –d이다. 만일 –d가 추가선택으로 입력되면

d(-없이)는 options변수에 기억된다(print지령에 대한 –R추가선택은 print 문자렬중의 첫 문자가 거꿀빗선(\ )이 되게 한다.).

3. 적합한 추가선택중의 하나가 +d이다. 만일 +d가 추가선택으로 입력되면 d(+기호와 함께)는 options변수안에 기억된다.
4. –d추가선택은 opts3에 대한 적합한 추가선택이다.
5. +d추가선택은 또한 opts3에 대한 적합한 추가선택이다.
6. –e추가선택은 적합하지 않다. 만약 추가선택이 적합하지 않으면 물음표(?)가 options에 기억된다. 적합하지 않은 인수가 OPTARG에 기억된다.
7. 추가선택앞에는 -도 아니고 +도 아닌것이 있다. getopts지령은 추가선택으로 처리하지 않으며 0 아닌 탈퇴상태를 되돌린다. While순환이 끝난다.

실례 10-140

(The Script)

```
    #!/bin/ksh
    # Program opts 4
    # Using getopts — Fourth try --
1.  alias USAGE='print "usage: opts4 [-x] filename " >&2'
2.  while getopts :x: arguments
    do
    case $arguments in
3.         x) print "$OPTARG is the name of the argument ";;
4.      :) print "Please enter an argument after the -x option" >&2 USAGE ; ;
5.       \ ?) print "$OPTARG is not a valid option." >&2 USAGE; ;
        esac
6.  print "$OPTIND" # The number of the next argument to be processed
    done
```

(The Command Line)

```
7.  $ opts4 -x
    Please enter an argument after the -x option
    usage: opts4 [~x] filename
        2
8.  $ opts4 -x filex
        filex is the name of the argument
        3
9.  $ opts4 -d
     d is not a valid option.
     usage: opts4 [-x] filename
        1
```

설명

1. getopts가 실패하면 별명 USAGE에 인쇄할 진단오유통보문을 대입한다.
2. while 지령은 getopts의 탈퇴상태를 검사한다. 만일 getopts가 인수를 성공적으로 처리할수 있다면 탈퇴상태 0을 되돌리며 while 순환본체에 들어 간다. x추가선택앞에 있는 두점은 사용자가 적합하지 않는 추가선택을 입력하면 오유통보문을 현시하지 않도록 getopts에 알린다. x추가선택에 첨부된 두점은 x추가선택다음에 인수가 있다는것을 getopts에 알린다. 만일 추가선택이 인수를 가지고 있으면 인수는 getopts의 내부변수 OPTARG에 기억된다.
3. x추가선택을 인수에 주면 인수는 OPTARG변수에 기억되고 출력된다.
4. 인수가 x추가선택다음에 제공되지 않으면 두점은 변수 arguments에 기억된다. 적당한 오유통보문이 현시된다.
5. 만일 적당하지 않은 추가선택이 입력되면 물음표(?)가 변수 arguments에 기억되며 오유통보문이 현시된다.
6. 특수한 getopts변수 OPTIND는 처리하려는 다음추가선택의 번호를 유지한다. 그 값은 항상 지령인수의 실제값보다 하나 크다.
7. x추가선택은 인수를 요구한다. 오유통보문이 출력된다.
8. 인수이름은 filex이다. 변수 OPTARG는 인수 filex의 이름을 유지한다.
9. 추가선택 d가 적합하지 않다. 사용통보문이 현시된다.

16. 보안

특권을 가진 스크립트 스크립트는 Korn쉘이 –p추가선택을 가지고 호출될 때 특권을 가진다. 특권추가선택을 사용하고 실제적인 UID와 또는 GID가 유효한 UID나 GID와 같지 않을 때 .profile은 실행되지 않으며 /etc/suid_profile 이라는 체계파일이 ENV파일대신에 실행된다.

제한된 쉘 Korn 쉘이 –r추가선택을 가지고 호출될 때 쉘의 기능은 제한된다. 쉘의 기능이 제한되면 cd지령은 사용할수 없으며 SEHLL, ENV, PATH변수들은 변경시킬수 없거나 해제할수 없다. 만약 첫번째 문자가 거꿀빗선 (\ )이면 지령들이 실행될수 없으며 방향바꾸기연산자(>, <, |, >>)들은 적합하지 않게 된다. 이 추가선택들은 set지령으로 해제 혹은 설정할수 없다. rksh지령은 제한된 쉘로 유지된다.

17. 내부지령

Korn쉘은 표 10-34에서 보여 준것과 같은 내부지령들을 가진다.

표 10-34. 내부지령과 그의 기능

지 령	기 능
:	아무런 동작도 하지 않는 지령: 탈퇴상태 0을 되돌린다.
. file	점지령을 읽고 파일로부터 지령을 실행한다.
Break	순환에서 보여 준다.

(표계속)

지 령	기 능
Continue	순환에서 보여 준다.
cd	등록부를 변경한다.
echo [args]	인수를 현시한다.
eval command	쉘은 실행하기전에 지령행을 두번 조사한다.
exec command	이 쉘의 위치에서 지령을 실행한다.
exit [n]	상태 n으로 쉘을 탈퇴한다.
export [var]	첨수로 알려 진 var를 작성한다.
fc -e [editor] [lnr] first last	리력목록에서 지령을 편집하기 위해 사용한다. 편집기가 규정되지 않으면 FCEDIT의 값이 사용된다. 만일 FCEDIT가 설정되지 않으면 /bin/ed/가 사용된다. 보통 리력은 fc –l이라는 별명을 가진다. 실례: fc –l　　리력목록에서 마지막 16개 지령들을 현시한다. fc -e emacs grep　　emacs편집기로 마지막 grep지령을 읽는다. fc 25 30　　FCEDIT에서 지정된 편집기(기정으로는 ed편집기)는 25부터 30까지의 지령들을 읽어 들인다. fc –e -　　마지막지령을 재실행한다. fc –e -Tom=Joe 28　　리력지령 28에서 Joe 로 Tom을 치환한다. fg　　마지막배경일감을 전경으로 보낸다. fg %n　　일감번호 n을 전경으로 보낸다. 정확한 일감번호를 찾기 위해 일감들을 현시한다.
jobs [-l]	활동적인 일감들을 PID번호와 –l추가선택, 번호를 가지고 표시한다. 실례: $jobs [3]+Running　　sleep 50& [1]-Stopped　　vi [2] Running　　sleep%
kill [-signal process]	프로쎄스의 PID번호 혹은 일감번호로 신호를 보낸다. 신호에 대한 목록을 /usr/include/signal.h에서 보여 준다. 신호들: SIGHUP1　　/* 정지 (련결하지 않는다) */ SIGINT2　　/* 새치기 */ SIGQUIT3　　/* 탈퇴 */ SIGILL4　　/*적합치 않은 명령문(사용될 때 재설정되지 않는다.) SIGTRAP5　　/* 추적 (사용될 때 재설정되지 않는다) */ SIGIOT6　　/* IOT 명령문 */ SIGABRT6　　/* 중지에 의해 사용되며 SIGIOT를 치환한다 */

(표계속)

지 령	기 능
	SIGEMT7 /* EMT 명령문 */
	SIGFLB8 /* 류동소수점 제외 */
	SIGKILL9 /* 없앤다(사용하거나 무시할수 없다.) */
	SIGBUS10 /* 모선오유 */
	SIGSEGV11 /* 토막위반 */
	SIGSYS12 /* 체계호출에 대한 틀린 인수*/
	SIGPIPE13 /*읽기 위해 하나가 아닌 파이프우에서 쓴다.*/
	SIGALRM14 /* 경보 박자 */
	SIGTERM15 /* 삭제로부터 소프트웨어완결 신호*/
	SIGURG16 /* I/O 통로에서의 긴급한 조건 */
	SIGSTOP17 /* tty로부터 정지신호를 전송가능한 신호 */
	SIGTSTP18 /* tty로부터가 아닌 보낼수 있는 정지신호 */
	SIGCONT19 /* 정지된 프로쎄스를 계속 */
	(Kill지령과 신호이름을 사용하여 SIG앞붙이를 없애고 –기호와 함께 신호이름을 앞에 놓는다.)
	실례;
	Kill – INT %3
	Kill – HUP 1256
	Kill – 9%3
	Kill %1
getopts	지령행을 해석하기 위해 쉘스크립트에서 사용하며 적당한 추가선택인가를 검사한다.
hash	모든 추적된 별명들을 현시한다.
login[username]	
newgrp[arg]	실지 group ID를 group ID로 변환한다.
print – [nrRsup]	echo에 위해 치환한다. print에서 보여 준다.
pwd	현재작업등록기를 출력한다.
read[var]	표준입력으로부터 변수 Var에 행을 읽어 넣는다.
readonly[var]	변수 Var를 읽기전용으로 작성한다.
return[n]	함수에 주어 진 값을 되돌린다.
Set [-aefhknoptuvx –[-o option] [-A arrayname][arg]	
	실례;
	Set 모든 변수들과 그 값을 현시한다.
	Set + 값이 없이 모든 변수들을 현시한다.
	Set –o 모든 추가선택설정들을 현시한다.
	Set a b c 위치파라메터 $1, $2, $3 등을 재설정한다.

(표계속)

지 령	기 능	
	Set –s	$1, $2, $3 은 자모별로 분류한다.
	Set – o vi	vi선택을 설정한다.
	Set – xv	오유수정하기 위해서 xtrace와 verbose 추가선택을 가능하게 한다. Set - - 모든 위치파라메터들을 해제한다.
	Set - - "$x"	지어 x가 –x이라도 $1을 x의 값으로 설정한다.
	Set = = $x	x안에 있는 매개 항목들의 경로이름을 확장시키고 매개 항목에 위치파라메터들을 설정한다.
	Set – A name tom dick harry	이름[0]을 tom으로 설정한다. 이름[1]을 dick으로 설정한다. 이름[2]를 harry로 설정한다.
	Set +A name joe	이름[0]은 joe로 재설정되고 배렬의 나머지는 그대로 남아 있는다. 이름[1]은 dick이다. 이름[2]는 harry이다.
	(추가선택을 설정하기 위해 –o 기발을 사용한다. 추가선택을 해제하기 위해 +o기발을 사용한다.)	
	실례:	
	set – o ignorreof	
	추가선택:	
	allexport	이것을 설정한 다음에 정의되거나 혹은 변경된 모든 변수들을 반출한다.
	bgince	보다 더 적은 우선권을 가진 배경일감을 실행시킨다. nice 대신에 사용된다.
	emacs	emacs내부편집기를 설정한다.
	errexit	쉘은 지령이 0 아닌 탈퇴 상태로 되돌릴 때 탈퇴된다.
	gmacs	내부gmacs편집기를 설정한다.
	ignoreeof	쉘완료로부터 EOF(Coontrol-D)건을 무시한다. 탈퇴하기 위해서는 exit를 사용해야 한다.
	keyword	쉘환경에 지령행의 임의의 곳에서 발생하는 keyword인수들을 첨가한다.
	markdirs	파일이름확장으로부터 모든 등록부이름들의 결과에서 \ 을 끌어넣기한다.
	monitor	일감조종을 설정한다.
	noclobber	방향바꾸기연산자 >를 사용하여 파일들의 겹쳐쓰기를 방지한다. 겹쳐쓰기를 하기 위해서는 >\ 를 사용한다.
	noexec	ksh –n과 같다. 지령들을 읽지만 실행은 하지 않는다. 쉘스크립트에서 문법오유를 검사하기 위해 사용한다.

(표계속)

지 령	기 능	
	noglob	ksh통용기호메타문자들로 경로이름확장을 무효하게 한다.
	nounset	만일 변수가 설정되지 않았다면 배경오유를 현시한다.
	privileged	설정된 uid프로그람을 특권을 가진 방식에서 돌린다.
	trackall	ksh는 추적된 별명으로 되게 하기 위해 매 지령을 자동적으로 대화형쉘들에서 돌린다.
	verbose	보통 오유수정에서 표준오유에 매 입력행을 현시한다.
	vi	vi내부편집기를 설정한다.
	viraw	시간입력에서 vi 문자를 규정한다.
	xtrace	매 지령을 확장하고 PS4재촉문에서 현시된다.
shift [n]	왼쪽으로 n번 위치파라메터를 밀기한다.	
times	이 쉘로부터 프로쎄스를 위해 축적된 사용자와 체계시간을 현시한다.	
trap[arg][n]	쉘이 신호 n(0, 1, 2 혹은 15)를 수신할 때 arg를 실행한다.	
type[command]	지령의 형태를 현시한다. 실례로 pwd는 내부쉘이다. ksh에서 whence –v를 위한 별명이다.	
typeset[option][var]	쉘변수와 함수를 위한 속성과 값을 설정한다.	
ulimit[option size]	프로쎄스에서 최대한계를 설정한다.	
	실례:	
	ulimit –a	모든 한계를 현시한다.
		시간(s)은 한계가 없다.
		파일(블로크)은 한계가 없다.
		자료(kbyte)524280
		탄창(kbyte)8192
		기억기(kbyte)한계가 없다.
		Coredump(블로크)는 한계가 없다.
	다른 추가선택:	
	-c 크기	블로크크기로 coredump를 제한한다.
	-d 크기	블로크크기로 자료크기(실행할수 있는)를 제한한다.
	-f 크기	블로크크기(기정)로 파일의 크기를 제한한다.
	-m 크기	kbyte의 크기로 물리기억기의 크기를 제한한다.
	-s 크기	kbyte의 크기로 탄창구역의 크기를 제한한다.
	-t 초수	초수로 프로쎄스 실행시간을 제한한다.
umask [mask]	인수가 없이 허가를 위한 파일작성마스크를 출력한다.	
umask [octal digits]	자기자체, 그룹 그리고 기타 등등을 위한 사용자파일작성방식마스크	

(표계속)

지 령	기 능	
unset [name]	변수나 함수의 값을 해제한다.	
wait [pid#n]	PID번호, n을 가진 배경프로쎄스를 대기하며 완결상태를 알린다.	
whence [command]	지령에 대한 통보를 현시한다.	
	실례 :	
	whence –v happy	happy는 함수이다.
	whence –v addon	addon은 비정의함수이다.
	whence –v ls	ls는 \ bin/ls을 위한 추적된 별명이다.
	whence ls	/bin/ls

18. Korn쉘호출인수

Korn쉘을 호출할 때 그의 특성을 조종하기 위해 추가선택들을 가질수 있다. 표 10-35에서 보여 준다.

표 10-35. Ksh에서 인수

지 령	기 능
-a	자동적으로 모든 변수들을 반출한다.
-c cmd	지령문자렬을 실행한다.
-e	지령이 령이 아닌 상태로 되돌이할 때 탈퇴한다.
-f	파일이름메타문자확장을 해제한다.
-h	추적된 별명으로 취급하도록 지령을 처리한다.
-i	대화형방식을 설정한다.
-k	예약어추가선택을 설정한다. 지령에서 모든 건인수들은 일부 환경을 만들게 한다.
-m	분리된 프로쎄스그룹에서 실행하기 위해 배경안에 실행된 지령을 넣으며 Control – C 혹은 logout가 나타나면 실행을 계속한다. 일감이 완결되였다는 통보문을 보낸다.
-n	오유수정을 위해 사용될수 있다. 지령들이 조사되지만 실행되지는 않는다. –x와 –v추가선택과 함께 사용될수 있다.
-o	set지령의 표에서 목록화된 이름에 의하여 설정될수 있는 추가선택들을 허용한다.
-p	특수한 방식을 설정한다. Setuid프로그람을 실행하기 위해 사용한다.
-r	제한된 방식을 설정한다.
-s	stdin기정으로부터 지령을 읽는다.
-t	쉘입력에서 발견한 첫번째 지령을 실행한 다음 탈퇴하기 위해 쉘을 발생하고 – c 추가선택을 규정한다.

(표계속)

지 령	기 능
-u	해제된 변수의 어떤 참조는 오유를 고려한다.
-v	스크립트 혹은 표준입력의 매행은 임의의 구문해석전에 현시되며 변수바꿔넣기 혹은 다른 프로쎄스를 실행한다. 출력은 표준오유에 써진다. 오유수정을 위해 사용된다.
-x	스크립트 혹은 표준입력의 매행은 실행되기전에 현시된다. 파일이름확장, 변수바꿔넣기 그리고 지령바꿔넣기는 출력에 나타난다. 모든 출력은 PS4재촉문, 기호 +, 공백의 값과 함께 처리된다. 행이 표준오유에 넣어 진다.

KORN쉘에 대한 련습

련습문제 28: 기동

1. 어떤 쉘을 사용하는가? 어떻게 알수 있는가?
2. 홈등록부에 .profile 혹은 .kshrc파일을 가지고 있는가? .profile 과 .kshrc 파일사이에 어떤 차이가 있는가? ENV파일은 무엇이며 그것을 변경하면 어떻게 효력을 나타낼수 있는가?
3. 기정의 1차재촉문은 무엇인가? 기정의 2차재촉문은 무엇인가? 지령행에서 1차재촉문을 변경하고 사용자등록파일이름을 포함하시오.
4. 다음과 같이 매 변수를 설정하는 목적은 무엇인가?

   ㄱ. Set -o ignoreeaf
   ㄴ. Set -o noclobber
   ㄷ. Set -o trackall
   ㄹ. Set -o monitor
   ㅁ. Set -o vi

   ENV파일에서 왜 이 변수들을 설정하는가? PATH의 목적은 무엇인가? PATH변수의 원소는 무엇인가?
5. 국부변수와 환경변수의 차이는 무엇인가? 모든 변수를 어떻게 목록화하는가? 환경변수들만을 어떻게 현시하는가? 현재 모든 추가선택설정을 현시하기 위해서는 다음과 같이 입력한다.

   Set –0

   어느 추가선택설정이 돌아 가는가?
6. 완전한 이름을 포함하는 myname이라는 국부변수를 참조하시오. 현재변수를 반출하시오. 재촉문에 다음과 같이 입력하시오.

   ksh

   변수이름이 반출되였는가? 어미쉘로 돌아 가기 위해 exit를 입력하시오. 읽기전용인 변수 name을 만드시오. 읽기전용변수란 무엇인가?

7. 보통 사용되는 위치파라메터는 무엇인가? 다음과 같이 입력하시오.

```
set apples pears peaches plums
```

위치파라메터를 사용하여 plums를 현시하시오.
apples peachs를 현시하시오.
Apples peaches pluches plums를 현시하시오.
파라메터번호를 현시하시오.
남새목록으로 위치파라메터를 재설정하시오.
모든 남새목록을 현시하시오. 파일목록은 어떻게 되는가?
다음과 같이 입력하시오.

```
Set - -
Print $ *
```

어떻게 동작하는가?

8. 현재쉘의 PID를 현시하시오. 재촉문에 다음과 같이 입력하시오.

```
grep $ LOGNAME /etc/passwd
echo $?
```

$?는 무엇을 하는가 말해 보시오. 지령의 실행에 대하여 탈퇴상태는 무엇을 하는가?

9. .profile에서 1차, 2차 2개의 재촉문을 변경하시오.등록해제함이 없이 .profile을 어떻게 재실행하며 다시 등록하는가?

련습문제 29: 리력

1. HISTSIZE변수는 무엇으로 설정되는가? HISTFILE변수는 무엇으로 설정되는가? set –o vi가 있는가를 보기 위해서 .kshrc파일을 검사하시오. 그것이 설정되여 있지 않으면 그것을 .kshrc 파일에 설정하고 다음과 같이 입력하여 파일을 재실행하시오.

```
.  . kshrc
```

2. 지령행에 다음의 지령들을 입력하시오.

```
ls
date
who
cal 2 1993
date +%T
```

리력 또는 fc –l 을 입력하시오. 이 지령들은 무엇을 수행하는가? 리력목록을 반대로 현시하시오. 번호없이 리력목록을 현시하시오. 현재지령과 그 앞의 5개 지령을 현시하시오. 열번째 명령으로부터 현재까지 모든것을 현시하시오. 제일 마지막 ls지령으로부터 마지막 cal지령까지의 모든것을 현시하시오.

3. r지령을 사용해서 마지막지령을 재실행하시오. 문자 d로 시작하는 마지막지령을 재실행하시오. 1897로 출력된 cal지령의 년도를 변경시키시오. 현재

시간을 찾기 위해 date지령 +%T를 변경시키시오.

4. 리력이 설정되면 지령행에서 ESC건을 누르고 리력목록을 우로 이동하기 위해 K건을 사용하시오. ls지령을 ls –alf로 변화시키고 그것을 재실행하시오.
5. FCEDIT변수가 env지령에 의해 설정되였는가를 검사하시오.
   그것이 설정되여 있지 않다면 지령행에 다음과 같이 입력하시오.

   export FCEDIT = vi

   지금 지령행에 다음과 같이 입력하시오.

   fc -1 -4

   무엇이 일어 나는가?
6. 행을 실행하지 않고 목록우에 넣기 위해서는 리력목록으로부터 어떻게 배치하는가?
7. 지령행에 다음과 같이 입력하시오.

   touch a1 a2 a3 apples bears balloons a4 a45

   표 10-2와 10-3에서 보여 준 리력 ESC순서를 사용하여 a로 시작되는 모든 파일들을 현시하시오.

   ㄱ. a로 시작되는 첫 파일을 현시하시오.
   ㄴ. a로 시작되는 모든 파일목록을 현시하시오.
   ㄷ. b로 시작되는 첫 파일을 현시하시오.
   ㄹ.지령과 그것의 설명을 현시하시오.
8. 지령행에 다음과 같이 입력하시오.
   print a b c d e
9. 리력 Esc underscore 지령을 사용하여 다음과 같이 지령을 변경하시오.
   print e
   리력 Esc underscore지령을 사용하여 첫번째 지령을 출력으로 변화시키시오.
   print c

련습문제 30: 별명과 함수

1. 어떤 지령이 현재 설정된 모든 별명들을 현시하는가?
2. 어떤 지령이 모든 추적별명들을 현시하는가?
3. 다음의 지령들을 위한 별명들을 작성하시오.

   date +%T
   history –n
   ls –alf
   rm –i
   cp –i
   print
4. 별명을 어떻게 넘기는가?
5. 다음과 같은 지령을 포함하는 함수를 작성하시오.

   ls –F

```
print   -n "The time is "
date +%T
print -n "Your present working directory is"
pwd
```

6. 함수를 실행하시오.
7. 인수를 넘기기 위해 위치파라메터를 리용하는 자체함수를 작성하시오.
8. 어떤 지령이 함수와 그의 정의를 현시하는가?
9. 몇가지 print추가선택을 사용하시오.

련습문제 31: 쉘메타문자

1. meta라고 부르는 등록부를 작성하시오. 등록부를 변경시키기 위해 cd지령을 사용하시오. 다음과 같은 파일들을 작성하기 위해서 touch를 사용하시오.
   abc abc1 abc2 abc2191 abc1 ab2 ab3 abc29 abc9 abc91 abc21xyz abc2121 noone nobody nothing nowhere
2. 다음의것을 하시오.
   ㄱ. 소문자 a로 시작하는 모든 파일들을 현시하시오.
   ㄴ. 대문자 A 다음에 2개 물음표로 시작하는 모든 파일을 현시하시오.
   ㄷ. 수자로 끝나는 모든 파일들을 현시하시오.
   ㄹ. abc 다음에 한개의 수자로 된 모든 파일들을 현시하시오.
   ㅁ. nothing 혹은 noone과 일치되는 모든 파일들을 현시하시오.
   ㅂ. abc다음에 한개이상의 수자로 된 모든 파일들을 현시하시오.
   ㅅ. 패턴 abc를 포함하지 않는 모든 파일들을 현시하시오.
   ㅇ. ab 다음에 3 또는 4를 포함하는 모든 파일들을 현시하시오.
   ㅈ. a 또는 A로 시작하여 그다음에 b 그리고 한개의 수자로 끝나는 모든 파일들을 현시하시오.
   ㅊ. 일치되는것이 없는 경우 어떤 오유통보문을 현시하는가?

련습문제 32: 물결표(~), 인용부호 그리고 지령바꿔넣기

1. 다음의것을 하기 위해 물결표를 사용하시오.
   ㄱ.사용자의 홈등록부를 현시하시오.
   ㄴ.사용자의 이웃등록부를 현시하시오.
   ㄷ.사용자의 이전작업등록부를 현시하시오.
   ㄹ.사용자의 현재작업등록부를 현시하시오.
2. 현재작업등록부의 값은 어떤 변수가 가지고 있는가? 이전의 작업등록부값은 어떤 변수가 가지고 있는가?
3. 사용자 이전작업등록부로 가기 위해 –를 사용하시오.
4. 다음의 출력을 현시장치에 보내기 위해 print지령을 사용하시오(< >안에 있는 단어는 확장될수 있는 변수이름이고 [ ]안에 있는 단어는 실행되는 지령의 출력이다. 즉 지령바꿔넣기를 리용한다.).

Hi <LOGNAME> how's your day going?
"No, <LOGNAME> you can't use the car tonight!" , she cried.
The time is [Sun Feb 21 13:19:27 PST 2001]
The name of this machine is [eagle] and
the time is [31:19:27]

5. 사용자이름목록을 포함하는 파일을 작성하시오. 지금 지령바꿔넣기를 사용하여 얻어 진 사용자이름목록을 포함하는 nlist라는 변수를 창조하시오.
   ㄱ. 변수의 값을 현시하시오. 지령바꿔넣기가 목록형식화에 어떤 영향을 주는가?
   ㄴ. ps-eaf명령의 출력으로 변수를 설정하여 이것을 검사하시오.
   ㄷ. 형식화가 어떻게 되는가?

련습문제 33: 방향바꾸기

1. 편집기에서 다음의 2개 행으로 된 ex6이라는 본문파일을 작성하시오.

   Last time I went to the beach I found a sea shell.
   While in Kansas I found a corn shell.

2. 지금 ex6파일에 다음의 《National Enquirer says someone gave birth to a shell, called the born shell》이라는 행을 추가하시오.
3. Ex6파일을 자기자신에게 전자우편으로 부치시오.
4. 파이프를 사용하여 ex6파일에서 행(wc-l)의 번호를 계수하시오.
5. 모든 설정된 추가선택들을 현시하기 위해 다음과 같이 입력하시오.

   set –o

   noclobber변수설정을 가지겠는가? 아니라면 다음과 같이 입력하시오.

   set –o noclobber

   어떤 일이 일어 나는가?
6. 지령행에 다음과 같이 입력하시오.

```
cat << FINIS
How are you $LOGNAME
The time is 'date' Bye!!
FINIS
```

   무엇이 현시되는가?
7. 타브들을 리용하여 해보시오.

```
cat <<- END
hellow there
how are you
END
```

   무엇이 현시되는가?
8. 지령행에 다음과 같이 입력하시오.

```
kat file 2> error || print kat failed
```

무엇이 일어 나는가? 왜?

9. 지령행에 다음과 같이 입력하시오.

    cat zombie 2> errorfile || print cat failed.

   무엇이 일어 나는가? 왜? &&연산자는 어떻게 동작하는가? 한개 지령으로 검사하시오.

10. 뿌리등록부아래서부터 a로 시작하는 모든 파일들을 현시하기 위해서 find지령을 사용하시오.

련습문제 34: 일감조종

1. 지령행에 다음과 같이 입력하시오.

    mail <user>Press control – z

   다음과 같이 입력하시오.

    jobs

   중괄호안에 있는 번호는 무엇인가?

2. 다음과 같이 입력하시오.

    sleep 300
    jobs
    bg

   bg는 무엇을 하는가? +와 -기호는 무엇을 가리키는가?

3. 일감조종을 사용해서 mail일감을 없애시오.
4. 편집기로 가시오. 일감을 중지하기 위해 ^Z를 누르시오. 중지된 vi일감을 전경으로 되돌리시오. 무슨 지령을 입력해야 하는가?
5. 다음지령을 입력하시오.

    jobs –1

   출력은 무엇인가?

6. TMOUT변수를 어떻게 사용하는가?
7. 다음지령을 실행할 때 핵심부에 의해 시간이 얼마나 걸리는가?

    (sleep 5; ps –eaf)

련습문제 35: info 쉘스크립트작성

1. info라는 프로그람을 작성하시오. 프로그람을 실행하기전에 chmod지령으로 실행가능한 프로그람을 작성하시오.
2. 프로그람은 해설을 포함해야 한다.
3. 프로그람은 실행할 때 다음과 같은것을 하여야 한다.

   ㄱ. 사용자등록가입번호를 출력하시오.
   ㄴ. 시간과 날자를 출력하시오.
   ㄷ. 현재작업등록부를 출력하시오.
   ㄹ. 어미등록부안에 있는 모든 등록부파일들을 현시하시오.
   ㅁ. 사용되고 있는 쉘이름을 현시하시오.

ㅂ. 등록가입이름을 포함하고 있는 암호파일로부터 한행을 현시하시오.
ㅅ. 사용자 ID를 현시하시오.
ㅇ. 이 콤퓨터이름을 현시하시오.
ㅈ. 디스크사용법을 현시하시오.
ㅊ. 이 달에 대한 달력을 현시하시오.
ㅋ. goodbye라고 하고 시간을 현시하시오.

련습문제 36: 보조문자렬의 변수확장

1. 다음의것을 수행하는 스크립트를 작성하시오.
   ㄱ. 홈등록부에 mypath라고 하는 변수를 설정하시오.
   ㄴ. mypath값을 현시하시오.
   ㄷ. mypath에 있는 경로의 마지막원소를 현시하시오.
   ㄹ. mypath에 있는 경로의 첫 원소를 현시하시오.
   ㅁ. 변수 mypath의 마지막원소를 제외한 모든것을 현시하시오.

련습문제 37: 스크립트검색

1. CD에 의해 제공되지 않으면 datafile이라는 파일을 작성하시오.
그것은 두점으로 분리된 마당들로 구성된다.
   ㄱ.이름과 별명
   ㄴ. 전화번호
   ㄷ. 주소 (거리, 도시, 구역, 그리고 우편번호)
   ㄹ. 생일(04/12/66)
   ㅁ. 로임
2. 파일에 10개의 실체를 넣으시오. 다음의것을 수행하는 lookup이라는 스크립트를 작성하시오.
   ㄱ. 사용자를 환영한다.
   ㄴ. datafile에 모든 사용자의 이름과 전화번호를 출력하시오.
   ㄷ. datafile의 행수를 출력하시오.
   ㄹ. 사용자에게 good bye라고 말하시오.

련습문제 38: typeset사용하기

1. 다음의 스크립트를 작성하시오.
   ㄱ. 사용자에게 이름과 성을 입력하도록 요구하시오.
   ㄴ. 2개 변수에 대답을 기억시키시오.
   ㄷ. 새로운 ksh의 read지령을 사용하시오.
2. 처음과 마지막이름변수를 모두 소문자로 변환하기 위해 typeset지령을 사용하시오.

3. 사람이름이 tom jones인가를 검사하시오. 이름이 같으면 Welcome, Tom Jones를 현시하시오. 아니면 《Are you happy today, FIRSTNAME LASTNAME?》을 현시하시오(사용자의 처음과 마지막이름은 대문자로 전환된다.).
4. 물음에 대하여 사용자들이 대답을 입력하도록 하고 새로운 ksh의 test지령을 리용하여 대답이 yes인가 no인가를 검사하시오. yes이면 스크립트가 사용자에게 좋을 말을 하게 하고 no이면 사용자에게 집으로 가라고 하고 현재시간을 알려 준다.
5. lookup스크립트를 다시 작성하시오.
   ㄱ. 이 스크립트는 datafile에 입력을 추가하려고 하는가를 사용자에게 묻는다.
   ㄴ. Yes 혹은 y로 대답하면 다음의 입력을 묻는다.

      이름
      전화번호
      주소
      생일
      로임

   매 항목에 대한 변수에는 사용자입력이 대입된다.

실례

```
print –n “What is the name of the person you areadding to the file?”
read name The information will be appended to the datafile.
```

련습문제 39: if/else명령과 let지령

1. 시험에서 사용자에게 점수를 묻는 grades라는 스크립트를 작성하시오.
   ㄱ. 스크립트는 0으로부터 100까지의 가능한 점수범위에 있는가를 검사한다.
   ㄴ. 스크립트는 사용자가 A, B, C, D 또는 F를 맞았는가를 물어 본다.
2. 간단한 전자수판기능을 실행하는 calc라는 스크립트를 작성하시오. 스크립트는 간단한 차림표를 제공한다.

   [a]Add
   [s]Subtract
   [m]Multiply
   [d]Divide
   [r]Remainder

3. 사용자는 차림표로부터 한개 문자를 선택한다.
4. 사용자에게 0으로부터 100사이에 있는 2개의 옹근수를 입력하도록 요구한다.
5. 수자가 범위를 초과하면 오유통보문이 현시되고 스크립트는 탈퇴한다.
6. 프로그람은 2개의 옹근수로 산수연산을 진행한다.
7. 답은 10, 8 그리고 16진수로 현시된다.

련습문제 40: case명령문

1. 다음과 같은 스크립트 timegreet를 작성하시오.
   ㄱ. 사용자이름, 날자, 프로그람의 목적을 스크립트의 머리부에 설명부분으로 제공하시오.
   ㄴ. case명령문을 사용하여 다음과 같은 프로그람을 변경시키시오.

```
# The timegreet script by Ellie Quigley
you=$LOGNAME
hour='date | awk ' {print substr ($4, 1, 2)}' '
print "The time is: $(date)"
if (( hour > 0 && $hour < 12 ))
then
  print "Good morning, $you!"
  elif (( hour ==12 ))
  then
  print "Lunch time!"
  elif (( hour > 12 && $hour < 16 ))
  then
  print "Good afternoon, $you!"
  else
  print "Good night, $you!"
fi
```

2. if/else명령을 case명령으로 바꾸어 lookup스크립트를 다시 작성하시오. 한개 이상의 차림표항목을 첨가하시오.
   1) Add Entry
   2) Delete Entry
   3) Update Entry
   4) View Entry
   5) Exit

련습문제 41: select 순환

1. 다음의 스크립트를 작성하시오.
   ㄱ. 사용자이름, 날자, 프로그람의 목적을 스크립트의 머리부에 해설부분으로 넣으시오.
   ㄴ. 식료품차림표를 제공하기 위해 select순환을 사용하시오.

```
$ 음식
1) 불고기와 감자
```

2) 물고기와 감자튀기
3) 국과 쌀라드
선택하시오.1
푸짐한 식사
콜레스테롤에 주의를 돌리시오.
많이 드시오.

1) 불고기와 감자
2) 물고기와 감자튀기
3) 국과 쌀라드
선택하시오.2
영국료리
많이 드시오.

1) 불고기와 감자
2) 물고기와 감자튀기
3) 국과 쌀라드
선택하시오.3
건강식품들
치료식사는 싫증난다.
많이 드시오.

$음식
1) 불고기와 감자
2) 물고기와 감자튀기
3) 국과 쌀라드
선택하시오.5

2. 주차림표와 보조차림표를 작성하기 위해 select명령을 사용하여 lookup스크립트를 다시 작성하시오. 차림표는 다음과 같이 구성된다.

1) 입력추가
2) 입력지우기
3) 입력갱신
4) 입력보기
   ㄱ) 이름
   ㄴ) 전화번호
   ㄷ) 주소
   ㄹ) 생년월일
   ㅁ) 로임
5) 탈퇴

련습문제 42: 함수를 자동적재하기

함수를 자동적재하기 위한 단계:

1. myfunction이라는 등록부를 작성한다.
2. myfunction에로 등록부를 변화시키고 goodbye라는 파일을 작성하기 위해 편집기를 사용하시오.
3. goodbye라고 부르는 함수를 파일이름과 같은 goodbye파일안에 삽입하시오.
4. goodbye함수는 다음과 같다.

```
function good bye {
print The current time is $ (date)
print "The name of this script is $0"
print See you later $1
print Your machine is 'uname –n'
}
```

5. 편집기를 작성하고 탈퇴한다. 사용자는 같은 이름으로 된 함수를 포함하는 파일을 가진다.
6. 홈등록부로 가시오. 다음과 같은 행을 입력하여 편집기에서 .kshrc파일을 변경시키시오.

```
FPATH=$HOME\ myfunctions
```

7. 편집기를 탈퇴하고 현재환경에서 .kshrc를 실행하기 위해 dot지령을 사용하시오.
8. 이미 씌여 진 timegreet 스크립트에서 다음의 행들을 포함하시오.

```
autoload goodbye
goodbye $LOGNAME
```

9. timegreet스크립트를 실행하시오. goodbye함수의 출력이 나타난다.
10. lookup스크립트에서 매 차림표항목들을 위한 함수를 작성하시오. My functions이라는 등록부에서 lookup_ functions이라는 파일안에 함수를 기억시키시오.
11. lookup스크립트에서 함수를 자동입력하고 대응하는 표식에 대하여 함수호출을 진행하시오.
12. trap지령을 리용하여 사용자가 옹근수값이 아닌 차림표선택값을 입력하면 trap지령이 화면에 오유를 현시하고 스크립트가 정확한 자료를 다시 입력하도록 사용자에게 요구하게 하시오.

제 11 장. 대화형bash쉘

제 1 절. 소 개

대화형쉘에서 표준입력, 표준출력 그리고 표준오유는 모두 하나의 말단에 련결되여 있다. Bourne Again(bash)쉘을 대화형으로 사용할 때에는 bash재촉문상태에서 UNIX 지령들을 입력하고 응답을 기다려야 한다. bash는 여러가지 내부지령과 리력, 별명, 파일 그리고 지령완성, 지령행편성 등과 같은 지령행지름길기능들을 제공한다. 몇가지 특징들은 표준UNIX Bourne쉘에 현재 있는것들이지만 GNU대상과제는 POSIX의 특징들을 포함하여 새로운 많은 기능들을 포함하도록 쉘을 확장하고 있다. bash 2.x에 많은 UNIX의 Korn쉘과 C쉘의 특징들이 포함되여 있으므로 bash쉘은 표준Borune쉘과 웃방향호환성을 보장하며 대화형과 프로그람작성준위의 두 측면에서 아주 유용한 쉘이다. UNIX사용자들에 대하여 bash쉘은 표준쉘들인 sh, csh, ksh와 맞먹는 기능들을 제공한다[1].

이 장에서는 지령행에서 bash가 어떻게 대화하며 작업환경을 관리하는가에 대하여 중점을 두고 해설한다. 사용자는 능률적인 작업환경을 작성하기 위해 지름기능들과 내부기능들을 리용하는 방법을 배우게 된다. 다음장에서 이에 대하여 더 구체적으로 해설한다. 이 방법들을 습득하면 사용자는 일상과제들을 자동화하고 정교한 스크립트들을 개발하여 자기에게 편리한 작업환경을 작성하는 bash쉘스크립트들을 작성할수 있으며 관리자인 경우에는 자기자신뿐아니라 사용자들의 전체 그룹을 위해 이렇게 할수 있다.

1. bash의 판본

Bourne Again쉘은 Brian Fox가 1988년 1월 10일에 제안한것인데 그다음에 Chet Ramey가 계속 발전시키고 있다. Chet Ramey는 현재 공식적으로 bash를 관리하면서 그 기능을 높이고 있다. Bash의 첫 판본은 0.99이다. 현재 사용하고 있는 판본은 2.04이다. 중요한 기능들은 판본 2.0에서 완성되였지만 이때의 조작체계판본은 1.14.7이였다. 모든 판본들은 GNU배포물을 제한없이 사용할수 있다. 사용자가 사용하는 판본을 보려면 bash지령에 추가선택 --version을 주거나 BASH_VERSION환경변수의 값을 출력하면 된다.

실례 11-1

```
$ bash  - - version
GNU bash, version 2.03.0(1)-release (sparc-sun-solaris)
Copyright 1998 Free Software Fundation, Inc.

$ echo $BASH_VERSION
2.03.0(1)-release
```

---

[1]. bash가 Linux플래트홈을 위한 기정쉘이지만 지금은 Solaris 8로 묶어 지고 있다.

2. 기동

만약 bash쉘이 등록가입쉘이면 쉘재촉문이 나타나기전의 프로쎄스의 사슬구조는 다음과 같다.[2]

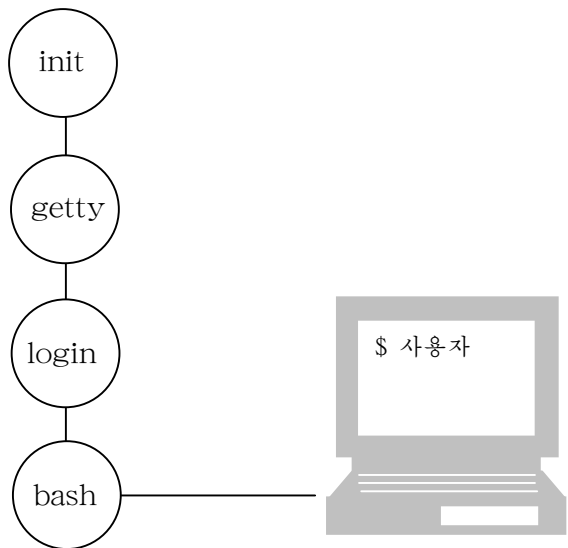



그림 11-1. bash 쉘의 기동

체계를 초기넣기할 때 실행되는 첫번째 프로쎄스는 init, PID #1이다. 그다음은 getty프로쎄스로 넘어 간다. 이 프로쎄스는 표준입력, 표준출력, 표준오유들을 위한 말단포구들을 대입하고 현시장치에 등록가입재촉문을 내보낸다. 그다음 /bin/login프로그람이 실행된다. login프로그람은 통과암호를 접수하고 비교하는 프로그람이다. 또한 초기환경을 대입하며 등록가입쉘인 /bin/bash를 기동하고 제일 마지막에 passwd파일안에 넣는다. bash프로쎄스는 /etc/profile체계파일들을 조사하고 지령들을 실행한다. 그다음 사용자의 홈등록부에서 초기화파일 .bash_profile을 조사한다. bash_profile로부터 지령들을 실행한 다음[3] .bashrc라는 사용자의 ENV파일로부터 지령을 실행하며 현시장치에는 기정으로 ($)재촉문이 현시되며 쉘은 지령대기상태에 들어 간다(초기파일에 대한 구체적인 내용은 558페지의 《환경》을 참고할것.).

지령행에서 쉘의 변경 사용자가 /etc/passwd파일을 변화시키지 않고 림시로 지령행으로부터 다른 쉘을 사용하려면 쉘의 이름을 건반으로 입력시켜야 한다. 실례로 현재 표준Bourne쉘을 사용하고 있는 상태에서 bash쉘로 바꾸려면 지령행에 건반으로 bash라고 입력하면 된다.

실례 11-2

```
1.  $ ps
    PID   TTY   TIME   CMD
```

2. bash 의 제일 최신판 판본을 얻기 위해서는 http://www.delorie.com/gnu/. 을 찾아 보아야 한다.

3. 그것들은 bash 에 의하여 사용되는 여러가지 초기화파일들로서 다음페지에서 언급한다.

1574 pts/6 0:00 sh

2. $ **bash**
   bash-2.03$

3. bash-2.03$ **ps**
   PID TTY TIME CMD
   1574 pts/6 0:00 sh
   1576 pts/6 0:00 bash

설명

1. ps지령의 출력은 현재 무슨 프로쎄스가 실행되고 있는가를 보여 준다. 실례에서 sh(Bourne쉘)가 실행중이다.
2. Bourne쉘재촉문이 나타난 상태에서 bash라고 건반으로 입력하면 Bourne Again쉘이 기동된다. 새로운 재촉문이 나타난다.
3. bash재촉문이 나타난 상태에서 ps지령을 실행한다. 결과는 2개의 쉘들이 실행중에 있으며 한개 쉘은 bash라는것을 보여 준다.

3. 환경

프로쎄스는 변수, 열기파일, 현재작업등록부, 함수, 제한자원, 신호 등으로 이루어 진다. 이것들은 임의의 쉘로부터 다른 쉘로 이어 지거나 작업환경의 구성과 같은 특징들을 정의한다. 사용자의 쉘구축은 쉘초기화파일들에서 정의된다.

초기화파일 bash쉘은 초기자료가 넣어 진 기동파일들을 가지고 있다. 파일들을 초기자료화하는것은 현재쉘의 일부를 파일에 모두 대입하여야 하기때문이다. 자식쉘은 참조되지 않는다(원천지령에 대하여서는 575페지의 《source 혹은 dot지령》에서 설명한다.). 초기화파일들은 쉘이 등록가입쉘이거나 대화형쉘(하지만 등록가입쉘은 아니다.) 또는 비대화형쉘(한개의 쉘스크립트)에 관련된다. 사용자가 등록가입할 때 쉘재촉문이 나타나기전에 체계쪽에서는 etc/profile이라는 초기화파일이 초기자료로 된다. 다음 그것이 존재하면 사용자의 홈등록부안에 있는 .bash_profile이 초기자료로 된다. 이것들은 사용자의 별명들과 함수들을 대입한 다음 사용자의 특수한 환경변수들과 기동스크립트들을 대입한다. 사용자의 .bash_profile이 없지만 .bash_login파일이 있으면 파일은 초기자료로 되며 .bash_login은 없지만 .profile이 있으면 초기자료로 될수도 있다(bash_login 파일은 C쉘의 .login파일과 같은것이고 .profile은 Bourne쉘을 기동할 때 보통 초기자료로 된다.). 다음의것은 bash가 초기화파일들을 처리하는 순서를 요약한것이다[4.](그림 11-2에서 보여 준다.).

```
if /etc/profile exists, source it,
    if ~/.bash_profile exists, source it,
        if ~/bashrc exists, source it,
    else if ~/.bash_login exists, source it,
    else if ~/.profile exists, source it.
```

4. ·쉘이 -noprofile 선택항목과 함께 호출되면 아무런 초기화파일도 읽지 않는다.

/etc/profile파일 /etc/profile 파일은 사용자가 등록가입할 때 과제를 실행하기 위해 체계관리자에 의해 대입되는 체계내부의 초기화파일이다. 이 파일은 bash쉘이 기동될 때 실행된다. 또한 모든 Bourne쉘과 Korn쉘사용자들에게 적용되며 /etc/motd 파일로부터 날자에 대한 통보를 현시하거나 새로운 전자우편을 위한 입출력완충기를 검사한다(다음의 실례들은 이 장에 대한 리해를 정확히 가진 다음 실행해야 한다.).

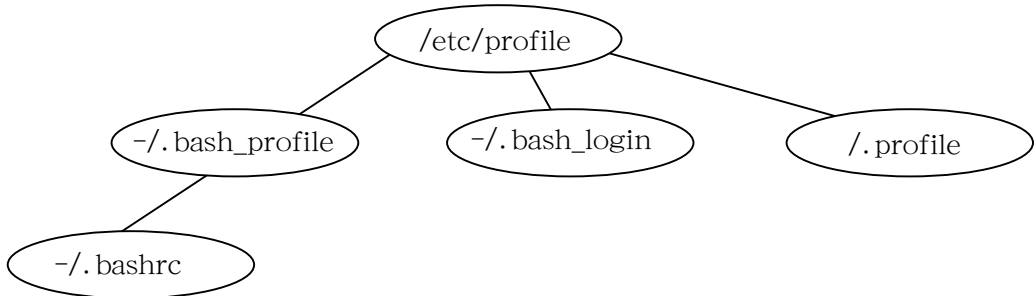



그림 11-2. 초기화파일들의 처리순서

실례 11-3

(Sample */etc/profile*)

```
# /etc/profile
# Systemwide environment and startup programs
# Functions and aliases go in /etc/bashrc
1.  PATH="$PATH:/usr/X11R6/bin"
2.  PS1="[\ u@\ h\ w]\ \ $ "
3.  ulimit –c 1000000
4.  if [ 'id –gn' = 'id –un' –a 'id –u' –gt 14 ]; then
5.      umask 002
    else
        umask 022
    fi
6.  USER='id –un'
7.  LOGNAME=$USER
8.  MAIL="/var/spool/mail/$USER"
9.  HOSTNAME='/bin/hostname'
10. HISTSIZE=1000
11. HISTFILESIZE=1000
12. export PATH PS1 HOSTNAME HISTSIZE HISTFILESIZE USER
    LOGNAME MAIL
13. for i in /etc/profile.d/*.sh ; do
14.     if [ -x $i ] ; then
15.             . $i
```

```
        fi
16. done
17. unset i #PATH
```

설명

1. PATH변수에 쉘이 지령을 탐색해야 하는 위치가 대입된다.
2. 1차재촉문이 대입된다. 그것은 사용자의 이름(\ u), @기호, 주기계(\ w) 그리고 화폐기호로 쉘창문안에 현시된다.
3. ulimit지령(쉘내부지령)은 파일의 최대크기를 1,000,000byte로 제한하도록 설정한다. 이러한 파일들은 자료를 중복시키지 않게 프로그람을 기억기에 배치하며 많은 자기원판구역을 차지하게 된다.
4. 이 행은 다음과 같다. 사용자의 그룹이름이 사용자이름과 같고 사용자의 ID번호가 14보다 크면...(5행을 참고).
5. umask를 002로 설정한다. 등록부가 창조될 때 그것들은 775허가를 얻으며 파일들은 664허가를 얻는다. 그렇지 않고 umask가 022로 설정되면 등록부에는 755, 파일에는 644허가가 주어 진다.
6. USER에 사용자이름(id−un)이 대입된다.
7. LOGNAME변수는 $USER의 값으로 설정된다.
8. MAIL변수는 사용자의 우편이 기억되는 우편스풀러에 대한 경로로 대입된다.
9. HOSTNAME변수는 사용자의 주콤퓨터이름으로 대입된다.
10. HISTSIZE변수는 1,000으로 설정된다. HISTSIZE는 재생되고(쉘기억기에 기억된 리력목록으로부터) 쉘이 탈퇴한후 리력파일에 기억되는 리력항목들의 번호를 조종한다.
11. HISTFILESIZE는 리력파일에 기억되는 지령들의 개수를 1000으로 제한하도록 설정된다. 즉 리력파일은 1000행후에 잘리워 진다(585페지의 《리력》을 참고).
12. 이 변수들은 부분쉘과 자식프로쎄스에서 리용될수 있도록 넘겨 진다.
13. etc/profile.d등록부에 있는 매 파일(.sh로 끝나는)에 대하여...(14행을 참고).
14. 파일이 실행형인가 검사해 보고 그렇다면...(15행을 참고).
15. 점지령으로 파일을 실행한다. Etc/profile.d등록부에 있는 파일들이 lang.sh와 mc.sh는 각각 문자와 폰트를 설정하고 Midnight Commande라고 하는 시각/열람기파일관리프로그람을 시동하는 mc함수를 창조한다. 파일관리자가 어떻게 동작하는가를 보려면 bash재촉문에서 mc를 입력하면 된다.
16. done예약어는 순환의 끝을 알려 준다.
17. 변수 i는 해제된다. 즉 쉘의 이름공간으로부터 소거된다. 변수 i의 값은 이전에 임의의 값으로 대입되였다면 for순환에서 대입된 값으로 된다.

~/.bash_profile 파일 ~/.bash_profile이 사용자의 홈등록부에 있으면 /etc_pro file 다음에 초기자료로 넣어 진다. ~/.bash_profile가 존재하지 않으면 bash는 다른 사용자정의 파일인 ~/.bash_login을 조사하고 그것을 초기자료로 한다. 만약 ~/.bash_ login이 존재하

지 않으면 ~/.profile을 초기자료로 하며 존재한다면 3개의 파일(~/.bash_profile, ~/.bash_login 혹은 ~/.profile)들중 어느 하나를 초기자료로 한다. bash는 또한 사용자가 .bashrc파일을 가지고 있거나 그것을 초기자료로 취급하면 검사한다.

실례 11-4

(Sample .bash_profile)

```
    # .bash_profile
    # The file is sourced by bash only when the user logs on.
    # Get the aliases and functions
1.  if [ -f ~/.bashrc ]; then
2.          . ~/.bashrc
    fi
    # User-specific environment and startup programs
3.  PATH=$PATH:$HOME/bin
4.  ENV=$HOME/.bashrc        # or BASH_ENV=$HOME/.bashrc
5.  USERNAME="root"
6.  export USERNAME ENV PATH
7.  mesg n
8.  if [ $TERM = linux ]
    then
          startx   # Start the X Window system
  fi
```

설명

1. 만약 사용자의 홈등록부안에 .bashrc파일이 있으면 2행에서 보여 준다.
2. 등록가입쉘을 위해 .bashrc를 실행한다.
3. PATH변수는 사용자의 bin등록부에 경로를 첨가한다. 보통은 쉘스크립트가 기억된 장소이다.
4. BASH_ENV┐(ENV)파일은 BASH_ENV(ENV)변수가 설정되여 있으면 대화형bash쉘과 스크립트들만을 초기자료로 하는 .bashrc파일의 경로이름이 설정된다. .bashrc파일은 사용자정의별명과 함수들로 이루어 진다.
5. 변수 USERNAME에 root가 설정된다.
6. 변수들이 반출되기때문에 자식쉘에서 효력이 있으며 다른 프로쎄스들은 그것들에 대하여 알게 된다.
7. mesg지령은 다른것들을 말단에 쓰지 못하도록 n추가선택과 함께 실행된다.
8. TERM변수의 값이 linux이면 startx는 리눅스유표창문안에서 대화형대화조종을 기동하는것이 아니라 X창문체계(다중가상유표를 쓸수 있는 도형사용자대면부)를 기동한다. 왜냐하면 ~/.bash_profile은 오직 등록가입할 때에만 초기

자료로 되기때문에 등록가입쉘은 X창문대화조종을 기동하게 된다.

ㄱ. BASH_ENV는 bash판본 2.0에서부터 사용된다.

BASH_ENV(ENV)변수 bash판본 2.0이전에는 BASH_ENV파일을 간단히 ENV파일이라고 하였다(Korn쉘에서도 같다.). BASH_ENV(ENV)변수는 ~/.bash_profile안에 설정된다. 이것은 대화형bash쉘이나 bash스크립트가 기동될 때마다 실행되는 파일의 이름을 대입한다. BASH_ENV(ENV)파일은 특수한 bash변수들과 별명들을 포함하고 있다. 원래이름은 .bashrc이지만 임의의것을 호출할수 있다. BASH_ENV(ENV)파일은 특수한 추가선택이 설정되거나(bash-p 혹은 set-0추가선택) - -norc지령행추가선택이 설정될 때(bash -norc 혹은 bash - -norc[bash 2.x+])에는 처리되지 않는다.

.bashrc파일 BASH_ENV(ENV)변수에 .bashrc이름(약속에 의해)을 대입한다. 이 파일은 자동적으로 새로 기동할 때나 대화형bash쉘이나 bash스크립트를 기동시킬 때에는 항상 초기자료로 된다. 그것은 오직 bash쉘과 관련되는 내용만을 포함한다.

실례 11-5

```
(Sample .bashrc)
# If the .bashrc file exists,it is in the user's home directory.
# It contains aliases (nicknames for commands) and user-defined
# functions.

# .bashrc
# User-specific aliases and functions

1.  set -o vi
2.  set –o noclobber
3.  set –o ignoreeof
4.  alias rm='rm –i'
    alias cp='cp –i'
    alias mv='mv –i'
5.  stty erase ^h
     # Soruce global definitions
6.  if [ -f /etc/bashrc ];then
     . /etc/bashrc
            fi
7.  case "$-" in
8.  *i*) echo This is an interactive bash shell
              ;;
9.          *)    echo This shell is noninteractive
                    ;;
       esac
10.  history_control=ignoredups
11.  function cd { builtin cd $1; echo $PWD; }
```

설명

1. set지령에 -o추가선택을 사용하면 특수한 내부추가선택을 설정 또는 해제할 수 있다. 추가선택이 –o이면 설정이고 추가선택이 +o이면 해제이다. vi추가선택은 대화형지령행편집기를 허용한다. 실례로 set -o vi는 대화형지령행편집기를 설정하며 set vi +o는 해제한다(표 11-1.)
2. noclobber추가선택은 방향바꾸기로 sont fielx>filex와 같은것을 사용할 때 파일의 겹쳐쓰기를 방지하기 위해 설정한다(11장 3절 9. 《표준 I\ O와 방향바꾸기》에서 보여 준다.).
3. 쉘에서 탈퇴하려면 보통 ^D(조종건-D)를 건반으로 입력시키면 된다. 또한 ignoreeof를 설정한 상태에서는 건반으로 exit라고 입력하면 된다.
4. rm, rm -i에 대한 별명은 대화형rm을 사용하자는것이다. 만약 실지로 파일들을 지우기전에 파일을 지우겠는가 하는 물음에 OK로 대답하면 사용자에게 다시 한번 확인하게 한다. cp, cp -I지령에 대한 별명은 대화형으로 복사를 처리한다.
5. stty지령을 삭제하는데 공백건을 사용한다.
6. /etc/bashrc파일이 있으면 그것을 초기자료로 한다.
7. 쉘이 대화형이면 특수한 변수인 $-는 i를 포함한다. 아니면 스크립트가 실행된다. case지령은 $-를 평가한다.
8. 만약 $-에 *i*가 대입되면 즉 문자렬이 한개의 i를 포함한다면 쉘은 “This is an interactive shell”을 현시한다.
9. 그외에 쉘은 “This shell is noninteractive”를 현시한다. 만약 스크립트를 기동하였거나 재촉문에 새로운 쉘이 있으면 쉘이 대화형인가 비대화형인가를 알수 있다. 여기에서는 다만 《대화형》쉘인가 《비대화형》쉘인가만을 검사한다.
10. history_control설정은 지령이 어떻게 리력파일에 기억되는가를 조종하는데 사용된다. 이 행은 《만약 리력파일속에 지령들이 이미 기억되여 있으면 또다시 기억하지 않는다.》는것을 알려 준다. 즉 2중복사를 무시한다.
11. 이것은 사용자정의함수이다. 사용자가 등록부들을 변경할 때 원래의 작업등록부인 PWD를 출력한다. 함수이름은 cd이며 지령 cd의 정의를 포함하고 있다. 특수한 내부지령인 builtin은 무한순환에 빠지는것을 방지하기 위해 cd지령앞에서 처리되여야 한다. 다시말하여 불명확한 호출에 주의해야 한다는것이다.

/etc/bashrc파일 체계내의 함수들과 별명들은 /etc/bashrc파일안에 설정할수 있다. 1차재촉문은 여기에서 설정된다.

실례 11-6

(Sample /etc/bashrc)

 # Systemwide functions and aliases
 # Environment stuff goes in /etc/profile

 # For some unknown reason bash refuses to inherit
 # PS1 in some circumstances that I can't figure out.
 # Putting PS1 here ensures that it gets loaded every time.

1. PS1="[\ u@\ h\ W]\ \ $ "
2. alias which="type –path"

설명

1. 체계내의 함수들과 별명들을 여기에서 설정한다. 원래의 bash재촉문은 사용자이름(\ u), @기호, 주기계(\ h), 현재작업등록부의 이름 그리고 화페기호($)로 설정된다(표 11—2에서 보여 준다.). 이 재촉문은 모든 대화형쉘들에서 나타난다.
2. 별명들 즉 지령들에 쓰이는 이름들은 항상 사용자의 .bashrc 파일안에 대입된다. 별명들을 다시 대입하면 bash가 초기기동될 때마다 효력을 가지게 된다. 사용자는 디스크에 있는 프로그람을 찾으려고 할 때 그것을 사용할수 있다. 다시말하면 어느 등록부에서 탐색한다는 구체적인 정보를 출력한다.

~/.profile파일 .profile파일은 사용자정의초기화파일이다.

이것은 사용자의 홈등록부에서 찾을수 있으며 sh(Bourne쉘)를 실행할 때는 오직 등록가입할 때에만 초기자료로 된다. Bash 를 실행할 때 우에서 언급한 임의의 초기화파일들을 찾을수 없으면 .profile을 실행한다. 사용자는 쉘환경을 전용화하거나 변경시킬수 있다. 환경과 말단들은 보통 여기에 설치되며 윈도우응용이나 자료기지응용이 초기화되면 여기서부터 시작된다.

실례 11-7

(Sample .profile)
A login initialization file sourced when running as sh or the
.bash_profile or .bash_login are not found.

1. TERM=xterm
2. HOSTNAME='uname –n'
3. EDITOR=/bin/vi
4. PATH=/bin:/usr/ucb:/usr/bin:/usr/local:/etc:/bin:/usr/bin:.
5. PS1="$HOSTNAME $ > "
6. export TERM HOSTNAME EDITOR PATH PS1
7. stty erase ^h
8. go () { cd $1; PS1='pwd'; PS1='basename $PS1'; }
9. trap '$HOME/.logout' EXIT
10. clear

설명

1. TREM변수에는 xterm지령의 말단값이 대입된다.

2. uname -n지령이 인용부호안에 있기때문에 쉘은 지령바꿔넣기를 한다. 즉 지령의 출력(주기계의 이름)이 HOSTNAME변수에 대입된다.
3. EDITOR변수에 /bin/vi를 대입한다. 전자우편 및 리력과 같은 프로그람들이 편집기를 정의할 때에는 이 변수가 효력이 없다.
4. PATH변수에는 쉘이 UNIX프로그람을 찾기 위해 검색하는 등록부입구점들을 대입한다. 실례로 ls지령을 주면 쉘은 등록부들가운데서 프로그람을 찾을 때까지 PATH를 검색한다. 찾지 못하면 쉘은 그 결과를 알려 준다.
5. 1차재촉문에는 HOSTNAME의 값인 콤퓨터이름, $ 그리고 >기호들이 대입된다.
6. 모든 변수목록이 반출된다. 그것들은 이 쉘로부터 기동된 자식프로쎄스에 의해 알게 된다.
7. stty지령은 말단추가선택들을 설정한다. 삭제건에는 ^H를 설정하며 공백건을 누르면 한문자가 삭제된다.
8. go함수를 정의한다. 이 함수는 등록부이름을 인수로 하여 현재의 작업등록부를 1차재촉문으로 설정한다.
   basename지령은 경로의 마지막입구점을 제외한 모든것을 지운다. 재촉문은 현재등록부를 가리킨다.
9. trap지령은 신호를 조종하는 지령이다. 사용자가 쉘에서 탈퇴할 때 등록해제되며 .logout파일이 실행된다. .logout파일은 등록탈퇴직전에 실행하는 지령들을 포함하고 있는 사용자정의파일이다. 이 지령들은 림시파일들을 삭제하거나 등록탈퇴시간을 기억한다.
10. clear지령은 현시장치를 지운다.

~/.bash_logout파일 사용자가 등록탈퇴(등록가입쉘을 탈퇴할 때)하려고 할 때 ~/.bash_logout파일이 있으면 이 파일은 초기자료로 된다. 이 파일은 보통 림시파일들을 지우고 리력파일들을 처리하며 등록탈퇴시간을 기록하기 위한 여러가지 지령들로 구성되여 있다.

기동파일들을 실행하지 않게 하는 추가선택 bash쉘이 –noprofile추가선택을 가지고 실행(bash - - noprofile)하면 /etc/profile, ~/.bash_profile, ~/.bash_login 혹은 ~/.profile과 같은 기동파일들은 초기자료로 사용되지 않는다. 또한 –p추가선택(bash –p로 줄 때)을 사용하면 bash는 사용자의 ~/.profile파일들을 읽지 못한다. Bash가 sh(Bourne 쉘)를 쓰려고 하면 가능한껏 Bourne 쉘의 특성을 가지도록 한다. 등록가입쉘은 오직 etc/profile과 ~/.profile을 초기자료로 한다. –noprofile추가선택은 이 특성이 불가능하게 한다. 쉘이 sh이면 임의의 다른 기동파일들을 초기자료로 할수 없다.

.inputrc파일 다른 기동초기화파일인 .inputrc파일도 bash가 기동될 때 읽어 진다. 이 파일이 사용자의 홈등록부에 있으면 건넣기특성과 문자렬묶음, 마크로, 조종기의 묶움을 전용화하는 변수들을 포함한다. 묶어 진 건들에 대한 이름과 기능에 대하여서는 본문을 편성하는 응용프로그람들에서 사용되는 서고를 보면 알수 있다. 이러한 묶음들은 지령행을 편성할 때 emacs와 vi편집기에서 효과적으로 사용된다(593페지의 《지령행편집》에서 보여 준다.).

4. 내부지령 set 와 shopt 에서의 bash 추가선택설정

set -0추가선택 set지령에 –o를 주면 추가선택들을 가질수 있다. 추가선택들은 쉘환경을 전용화한다. 거기에는 설정과 해제가 있으며 보통 BASH_ENV(ENV)파일안에 설정한다. set지령에 쓰이는 여러가지 추가선택들은 형식을 지름화하여 쓴다. 실례로 set -o nocl obber는 set -C로도 쓸수 있다(표 11-1에서 보여 준다.).

표 11-1. 내부set지령추가선택

추가선택이름	지름추가선택	의 미
allexport	-a	해제할 때까지 자동적으로 변수가 실행된 시각부터 반출하기 위해 새로운 변수들이나 변경된 변수들을 표식한다.
braceexpand	-B	대괄호확장을 쓸수 있다. 기정은 설정되여 있다.
emacs	-e	지령행편성에 Emacs내부편집기를 사용한다. 기정은 설정되여 있다.
eErrexit		지령이 0이 아닌 탈퇴상태값(실패)을 되돌려 주면 끝마친다. 초기화파일들을 읽을 때 설정되지 않는다.
histexpand	-H	리력바꿔넣기를 진행할 때 !와 !!를 쓸수 있다. 기정은 설정되여 있다.
history		지령행리력을 쓸수 있다. 기정은 설정되여 있다.
ignoreeof		쉘에서 탈퇴할 때 EOF(조종건-D)를 쓸수 없다. 건반으로 exit를 입력해야 한다. 쉘변수 IGNOREEOF=10과 같다.
keyword	-k	지령을 위해 환경안에 예약어인수를 넣는다.
interactive comments		대화형쉘들에서 지령행에 설명문을 줄 때 #지시자가 사용된다.
monitor	-m	일감조종을 허락한다.
noclobber	-C	방향바꾸기를 사용할 때 겹쳐쓰기로부터 파일을 보호한다.
noexec	-n	지령을 읽기만 하고 실행하지는 않는다. 스크립트 물음표를 검사하기 위해 사용한다. 대화형으로 실행될 때는 설정하지 않는다.
noglob	-d	경로이름의 확장을 불가능하게 한다.
notify	-b	배경일감을 끝마칠 때 사용자를 통보한다.
nounset	-u	설정되지 않은 변수를 확장하려고 할 때 오유를 현시한다.
onecmd	-t	한개의 지령을 읽고 실행한 다음에 끝마친다.
physical	-P	설정되였으면 cd나 pwd지령을 사용할 때 기호련결을 하지 않는다. 그대신 물리적인 등록부를 사용한다.

(표계속)

추가선택이름	지름추가선택	의 미
posix		쉘특성은 기정조작이 POSIX표준으로 되여 있지 않으면 변경된다.
privileged	-p	쉘은 설정되였으면 .profile이나 ENV파일을 읽지 않으며 쉘기능은 자동적으로 setuid스크립트들을 설정하는 환경으로부터 이어 지지 않는다.
posix		POSIX 1003.2로 기정특성을 변경한다.
verbose	-v	오유수정을 위해 확장방식을 설정한다.
vi		지령행편성에 vi내부편집기를 사용한다.
xtrace	-x	오유수정을 위해 echo방식을 설정한다.

형식

```
set -o 추가선택        #추가선택을  설정한다.
set +o 추가선택        #추가선택을 해제한다.
set -[a-z]          #추가선택을 지름화한다.–는 설정을 의미한다.
set +[a-z]          #추가선택을 지름화한다.+는 해제를 의미한다
```

실례 11-8

```
1.  set –o allexport
2.  set +o allexport
3.  set –a
4.  set +a
```

설명

1. allexport추가선택을 설정한다. 이 추가선택은 자동적으로 자식쉘에 모든 변수들을 대입한다.
2. allexport추가선택을 해제한다. 모든 변수들은 현재쉘에서 국부변수로 된다.
3. allexport추가선택을 설정한다. 1과 같다. 모든 추가선택들은 지름화되지 않는다(표 11-1에서 보여 준다.).
4. allexport추가선택을 해제한다. 2와 같다.

실례 11-9

```
1.  $ set –o
    braceexpand          on
    errexit              off
    hashall              on
    histexpand           on
    keyword              off
```

monitor *on*
noclobber *off*
noexec *off*
noglob *off*
notify *off*
nounset *off*
onecmd *off*
physical *off*
privileged *off*
verbose *off*
xtrace *off*
history *on*
ignoreeof *off*
interactive=comments *on*
posix *off*
emacs *off*
vi *on*

2. $ **set –o nolobber**
3. $ date > outfile
4. $ Is > outfile
   bash: outfile:Cannot clobber existing file
5. $ **set +0 noclobber**
6. $ Is > outfile
   7. $ **set –c**

설명

1. –o추가선택을 사용하면 set지령은 현재설정 혹은 해제된 모든 추가선택들을 현시한다.
2. 추가선택을 설정하기 위해서는 –o추가선택을 사용한다. noclobber추가선택은 대입되여 있다. 방향바꾸기를 사용할 때 파일우에 덧쓰기하지 않게 한다. noclobber가 설정되지 않을 때 >기호의 오른쪽 파일이 존재하면 잘라 지고 존재하지 않으면 만들어 진다.
3. UNIX date지령의 출력이 outfile에로 방향바꾸기가 진행된다.
4. 이때 outfile이 만들어 진다. Ls의 출력을 outfile로 대입하면 이미 있던 파일은 해당한 처리를 받는다.
5. set지령에 +o추가선택을 함께 쓰면 noclobber는 off로 된다.
6. 이때 noclobber가 대입되여 있지 않기때문에 outfile에로의 겹쳐쓰기를 할 수 있다.
7. set지령에 –C추가선택을 사용하는것은 noclobber를 ON으로 대입하기 위해서이다. +C는 off로 대입한다.

shopt내부지령(판본 2.x+) shopt내부지령은 대화형 bash의 새로운 판본들에서 set

지령처럼 사용한다. shopt는 set내부지령을 련결시켜 만드는데 쉘을 구축하기 위해 여러가지 추가선택을 추가할수 있다. 모든 shopt들에 대한 목록은 표 11-27을 보면 된다. 다음의 실례에서는 –p추가선택을 써서 사용할수 있는 모든 추가선택들을 현시한다. –u추가선택은 추가선택의 해제를 의미하며 –s추가선택은 현재 설정된것들중의 한개를 가리킨다.

실례 11-10

1. $ **shopt –p**
   shopt –u cdable_vars
   shopt –u cdspell
   shopt –u checkhash
   shopt –u checkwinsize
   shopt –s cmdhist
   shopt -u execfail
   shopt –s expand_aliases
   shopt –u extglob
   shopt –u histreedit
   shopt –u histappend
   shopt –u histverify
   shopt –s hostcomplete
   shopt –u huponexit
   shopt –s interactive_comments
   shopt –u lithist
   shopt –u mailwarn
   shopt –u nocaseglob
   shopt –u nullglob
   shopt –s promptvars
   shopt –u restricted_shell
   shopt –u shift_verbose
   shopt –s sourcepath
2. $ **shopt –s cdspell**
3. $ **shopt –p cdspell**
   shopt –s cdspell
4. $ **cd /home**
   /home
5. $ **pwd**
   /home
6. $ **cd /ur/lcal/ban**
   /usr/local/man

7. **$ shopt –u cdspell**
8. **$ shopt –p cdspell**
   shopt –u cdspell

설명

1. –p추가선택을 쓰면 shopt는 쉘추가선택의 모든 대입항목들과 설정(-s) 혹은 해제(-u)중의 현재값으로 상태를 현시한다.
2. –s추가선택을 쓰면 shopt추가선택으로 대입된다. cdspell추가선택은 cd지령에 사용한 등록부이름이 정확한가를 쉘에 물어 본다. 간단한 실례로 건반으로 typos라고 입력하면 빠진 문자들을 삽입하고 바꿔넣기도 하며 바로 잡기도 한다.
3. –p추가선택과 추가선택의 이름을 함께 사용하면 shopt는 추가선택이 설정되였는지 안되였는지를 가리킨다. 추가선택은 설정상태일 때 –s로 된다.
4. 실례로 홈등록부로 가기 위해 cd지령을 사용하였는데 그만 홈단어가 틀렸다. 쉘은 정확한 단어가 home이기때문에 hame이라는 단어를 처리한다. 등록부는 /home으로 변경된다.
5. pwd지령의 결과는 현재의 작업등록부를 현시하고 있다. 실례에서 보면 사용자가 틀리게 입력하였지만 정확히 변화되였다는것을 알수 있다.
6. 여기에서는 등록부이름이 틀렸는데 마지막단어 ban에 오유가 있다는것을 알수 있다. 쉘은 ban을 man으로 수정하고 틀린 문자들을 고침으로써 정확한 경로이름을 출력하도록 한다. ban에서 b는 첫번째 틀린 문자이므로 쉘은 an으로 끝나는 입구점들을 등록부에서 검사하여 man을 찾는다.
7. –u추가선택[ㄱ.]을 사용하면 shopt는 추가선택을 off로 대입한다.
8. –p추가선택과 추가선택이름을 함께 사용하면 shopt는 추가선택이 설정되였는지 안되였는지를 알려 준다. cdspell추가선택은 해제(-u)로 되여 있다.

ㄱ. 단어선택이나 추가선택들은 교환할수 있다. 그것들은 횡선을 포함하는 지령에서의 인수이다.

5. 재촉문

대화형쉘을 사용할 때 쉘은 입력을 위해 재촉문을 내보낸다. 재촉문이 나타나면 지령을 입력시킬수 있다는것을 의미한다. bash쉘은 4개의 재촉문을 제공하는데 첫번째가 화폐기호($)이고 두번째가 (>)기호이며 세번째와 네번째는 PS3과 PS4이다. PS3과 PS4는 후에 언급하도록 한다. 재촉문은 쉘이 대화형일 때 현시된다. 이 재촉문은 변경시킬수 있다. 변수 PS1에는 1차재촉문이 대입된다. 그러므로 화폐기호($)는 보통 UNIX지령을 입력하기 위해 등록가입되고 대기상태에 있을 때 나타난다. 변수 PS2는 2차재촉문인데 기호 (>)로 대입된다. 이 2차재촉문은 지령을 입력하고 Enter건을 누르면 나타난다. 1차재촉문과 2차재촉문은 변경시킬수 있다.

1차재촉문 화폐기호($)는 기정으로 첫번째 재촉문으로 대입되여 있다. 사용자는 이 재촉문을 변경시킬수 있다. 보통 재촉문은 /etc/bashrc안에 정의되여 있거나 사용자초기파일 .bash_profile 혹은 .profile(Bourne 쉘)안에 정의되여 있다.

실례 11-11

1. $ **PS1="$(uname –n) > "**
2. *chargers >*

설명

1. 첫번째 재촉문은 화폐기호($)이다. PS1재촉문은 콤퓨터ㄱ·의 이름으로 다시 대입되여 있는데 그것은 a>기호이다.
2. 새로운 재촉문이 현시된다.

ㄱ· uname n지령은 소괄호앞에 화폐기호가 있는 상태로 둘러 싸여 있기때문에 실행된다. 대신 거꿀인용부호안에 지령을 넣어야 한다(11장 3절 4.《지령바꿔넣기》에서 보여 준다.).

특수한 ESC코드를 재촉문으로 대입하기 재촉문에 특수한 ESC코드를 삽입하여 재촉문을 전용화할수 있다. 표 11-2에 특수문자들을 준다.

표 11-2. 재촉문문자설정

조종문자	결 과
\ t	현재시간을 HH:MM:SS 형식으로 준다.
\ d	날자를《요일 월 날자》 형식으로 준다(즉 Tue May 26).
\ n	행바꾸기
\ s	쉘의 이름
\ w	현재작업등록부
\ W	현재작업등록부의 토대이름
\ u	현재사용자이름
\ h	주기계이름
\ #	이 지령의 번호
\ !	이 지령의 리력번호
\ $	작용하는 UID가 0이면 #이고 그외에는 $이다.
\ nnn	8진수 nnn으로 응답하는 문자
\ \	\ 기호
\ [	문자를 출력할수 없는 코드의 시작. 재촉문안에 말단조종문자를 매몰하여 사용할수 있다.
\ ]	문자를 출력할수 없는 조종코드의 끝
bash version 2.x+에서 새로운 기능	
\ a	ASCII종울림문자
\ @	12시간 AM/PM 형식의 현재시간
\ H	주기계이름

(표계속)

조종문자	결 과
\ T	12시간 형식: HH:MM:SS으로 현재시간
\ e	ASCII ESC문자 (033)
\ v	Bash판본인데 례를 들면 2.03이다.
\ V	Bash의 발표와 통과수준인데 례를 들면 2.03.0

실례 11-12

1. $ **PS1="[\ u@\ h\ w] \ \ $ "**
   [ellie@homebound ellie]$
2. $ **PS1=" \ w: \ d> "**
   ellie:Tue May 18>

설명

1. 특수한 ESC코드를 사용하여 원래의 bash재촉문을 대입한다. \ u는 사용자의 등록가입이름을 주며 \ h는 주기계이름 그리고 \ w는 현재작업등록부이름을 준다. 여기에는 2개의 \ 기호가 있는데 첫번째 \ 기호는 두번째 \ 기호를 처리하는것으로서 결과는 \ $로 된다. 화페기호($)는 쉘프로쎄스에서 보호되며 그대로 현시된다.
2. 1차재촉문은 현재작업등록부의 이름을 주는 ESC코드 \ w와 오늘 날자를 주는 ESC코드 \ d로 처리된다.

2차재촉문 PS2변수에는 2차재촉문이 대입된다. 그 값은 기정으로 현시장치인 표준오유에 현시된다. 이 재촉문은 지령이 실행되지 않았거나 지나치게 많은 입력을 주었을 때 나타난다. 기정으로 2차재촉문에는 >가 대입된다.

실례 11-13

1. $ echo "Hello
2. **>** there"
3. hello
   there
4. $
5. $ **PS2="--→ "**
6. $ echo 'Hi
7. - - - - →
   - - - - →
   - - - - → there
   Hi

there

```
$
8.  $ PS2="\ s:PS2 > "
     $ echo 'Hello
          bash:PS2 > what are
          bash:PS2 > you
          bash:PS2 > trying to do?
          Bash:PS2 > '
          Hello
          What are
          you
          trying to do?
$
```

설명

1. 2중인용부호는 문자렬 "Hello"다음에 주어야 한다.
2. 새로운 행을 입력할 때 2차재촉문이 현시된다. 2중인용부호가 넣어 질 때까지 2차재촉문이 현시된다.
3. echo지령의 결과가 현시된다.
4. 1차재촉문이 현시된다.
5. 2차재촉문이 다시 대입된다.
6. 단일인용괄호는 문자렬 'Hi'다음에 주어야 한다.
7. 새로운 행에 입력될 때 2차재촉문이 나타난다. 단일인용괄호가 넣어 질 때까지 2차재촉문이 현시된다.
8. PS2재촉문은 쉘의 이름(\ s)다음에 두점, PS2 그리고 공백으로 이루어 진 문자렬로 대입된다.

탐색경로 bash는 지령행에 입력시킨 지령들을 처리하기 위해 PATH변수를 사용한다. path는 지령들을 검색할 때 쉘에서 사용하는 옹근두점으로 구분된 등록부목록이다. 기동경로는 체계에 따라 bash를 설치한 관리자에 의해 대입된다. 경로는 왼쪽에서부터 오른쪽으로 가면서 검사된다. path의 끝에 있는 점은 현재작업등록부를 현시한다. 지령이 경로안에 있는 등록부목록에서 찾지 못하면 쉘은 "filename:not found"라는 통보문을 표준오유에 출력한다. 경로는 bash쉘을 실행하면 보통 .bash_profile에서 설정되며 sh 즉 Bourne쉘을 리용하면 .profile파일에서 설정된다. 만약 경로내에 점이 포함되여 있지 않고 현재작업등록부로부터 지령을 실행한다면 스크립트의 이름은 ./program_name파 같이 ./기호를 앞에 놓아야 한다. 그래야 쉘이 프로그람을 찾을수 있다.

실례 11-14

(Printing the PATH)

```
1.  $ echo $PATH
    /usr/gnu/bin:/usr/local/bin:/usr/ucb:/bin:/usr/bin:.
```

(Setting the PATH)

2. $ **PATH=$HOME:/usr/ucb:/usr:/usr/bin:/usr/local/bin:**
3. $ **exprot PATH**
4. $ **runit**
   bash:runit:command not found
5. $ **./runit**
   <program starts running here >

설명

1. $PATH에 의해 분리된 PATH변수의 값이 현시된다. path는 옹근두점구분기호로 분리된 원소들로 구성되며 왼쪽에서부터 오른쪽으로 가면서 탐색된다. path의 끝에 있는 점은 사용자의 현재작업등록부를 가리킨다.
2. path를 대입하기 위해서는 옹근두점으로 구분된 등록부들의 목록을 PATH변수에 주어야 한다. 실례에서는 path의 끝에 점이 없으며 통보처럼 씌여진다.
3. path를 대입함으로써 자식프로쎄스는 경로들을 호출할수 있다. 개개의 행으로 PATH를 대입할 필요가 없이 한행에 쓸수 있다. 즉 다음과 같이 쓸수 있다.
   EXPORT path=$HOME:/usr/ucb:/bin: …
4. runit프로그람이 현재작업등록부에서 실행될 때 점이 검색경로안에 없기때문에 bash는 프로그람을 찾을수 없다.
5. 프로그람이름은 점과 빗선(./)이 함께 처리되기때문에 현재작업등록부에서 쉘은 프로그람을 찾을수 있다.

hash지령 hash지령은 쉘이 지령탐색에서 사용하는 내부하쉬표를 조종한다. 경로를 검사하지 않고 임의의 시각에 지령을 입력하려면 우선 지령을 건반으로 주며 쉘은 지령을 찾기 위해 경로검사를 한다. 그다음 기억기안에 있는 표에 지령을 기억시킨다. 다음번에 똑 같은 지령을 사용하려면 쉘은 지령을 찾기 위해 하쉬표를 사용한다. 이것은 목적한 경로를 검사하는데 걸리는 지령호출시간보다 매우 빠른 방법이다. 만약 자주 사용되는 지령이 있다면 하쉬표에 지령을 첨부할수 있다. 또한 하쉬표로부터 지령을 삭제할수도 있다. hash지령의 출력은 지령을 찾기 위해 표를 사용한 변수와 지령의 정확한 경로이름을 현시한다. hash지령에 –r추가선택을 함께 쓰면 하쉬표를 지운다. --기호는 인수들을 정지시키기 위한 검사추가선택을 무효로 한다. 하쉬표는 bash에 의해 자동적으로 제공한다.

실례 11-15

(Printing the PATH)

(Command line)

1. **Hash**
   hits *command*
   1. */usr/bin/mesg*
   4. */usr/bin/man*
   2. */bin/ls*
2. **hash –r**

3. **hash**
   No commands in hash table

4. **hash find**
   hits command
   0 /usr/bin/find

설명

1. hash지령은 등록가입대화조종에서 실행되는 정확한 경로이름을 현시한다(내부지령은 현시되지 않는다.). 수값은 지령을 찾기 위해 하쉬표를 사용한 회수이다.
2. hash지령에서 –r추가선택은 하쉬표에 있는 모든 기억된 위치값들을 지운다.
3. –r추가선택을 사용한 다음의 hash지령은 현재 표에 지령들이 없다는것을 알려 준다.
4. 자주 사용되는 지령들을 hash지령에 인수로 대입하면 하쉬표에 지령들을 첨부할수 있다. find지령이 침부되였다. 표는 지령이 아직 사용되지 않았기때문에 0으로 된다.

soruce 혹은 dot지령 source지령(C쉘로부터)은 bash쉘지령안에 내장되여 있다. dot지령(Bourne쉘로부터)은 source에 대한 다른 이름이다. 두 지령은 모두 인수로 스크립트이름을 가진다. 스크립트는 현재쉘환경속에서 실행되는데 즉 자식프로쎄스는 기동되지 않는다는것이다. 모든 변수들은 현재쉘환경의 한 부분으로 되는 스크립트안에 대입된다. source(혹은 dot)지령은 보통 초기화파일들인 .bash_profile, .profile이 변경되였을 때 그것들중 임의의것을 재실행하기 위해 사용된다. 실례로 EDITOR나 TERM변수들중 하나가 등록가입한 다음 .bash_profile안에서 변경되였다면 등록탈퇴함이 없이 .bash_profile안에 있는 지령을 다시 실행하고 그다음 등록가입하기 위해 source지령을 사용할수 있다. .bash_profile이나 임의의 쉘스크립트들은 점(.)이나 source지령들을 초기자료로 되게 할 필요가 없다.

실례 11-16

$ **source .bash_profile**
$ **. .bash_profile**

설명

source나 점(.)지령은 현재쉘의 초기화파일인 .bash_profile을 실행한다. 국부 또는 전역변수들은 이 쉘안에 재정의된다. 점지령은 등록탈퇴하지 않도록 파일이 변경된 다음에 다시 등록가입한다.ㄱ.

ㄱ. bash_profile 이 직접 스크립트로 실행되면 자식쉘이 기동된다. 다음변수는 자식쉘에서 설정된다. 자식쉘이 탈퇴할 때 어미쉘은 자식쉘에서 설정되여 있던 모든것을 가지지 못한다.

6. 지령행

등록가입한 다음 bash쉘은 화폐기호($)를 기정재촉문으로 현시한다. 쉘은 지령을 판단한다. 쉘이 대화형으로 실행되고 있을 때 말단으로부터 지령을 읽고 단어들로 지령을 줄수

있다. 지령행은 여러개의 공백이나 타브건에 의해 분류된 한개 또는 여러개의 단어로 이루어 지며 행바꾸기로 끝나고 Enter건을 누르면 처리된다. 첫번째 단어는 지령이며 그다음것들은 지령의 인수이다. 지령은 ls나 date와 같은 UNIX실행프로그람이여야 하며 사용자정의함수, cd나 pwd 혹은 스크립트와 같은 내부지령도 포함한다. 지령은 지령행을 해석할 때 쉘이 해석할수 있는 메타문자들로 구성된다. 만약 지령이 너무 길면 다음행에 계속하여 넣을수 있다. 2차재촉문은 지령행이 완결될 때까지 계속 현시되여 있다.

지령들의 처리순서 지령행의 첫번째 단어는 실행하려는 지령이다. 지령은 예약어, 별명, 함수, 특수한 내부지령들 혹은 그의 응용, 실행프로그람 혹은 쉘스크립트들로 구성된다.지령은 다음과 같은 순서로 분류하여 실행된다.

1. 별명
2. 예약어(실례로 if, function, white, until)
3. 함수
4. 내부지령
5. 실행할수 있는것과 스크립트

특수한 내부지령들과 함수들은 쉘내부에 정의되여 있으며 실행이 더욱 고속화되면서 현재쉘에서 실행된다. ls와 date와 같은 스크립트들과 실행프로그람들은 디스크에 기억되여 있으며 쉘이 지령들을 실행하려면 이것들을 우선 PATH환경변수검사를 위해 적당한 등록부안에 배치시켜야 한다. 그다음 쉘은 스크립트들을 실행하는 새로운 쉘을 준비한다.

사용자가 사용하는 지령의 형태 즉 내부지령, 별명, 함수 혹은 실행파일들을 알기 위해 type지령을 사용한다(실례 11-17).

실례 11-17

```
$ type pwd
pwd is a shell builtin
$ type test
test is a shell builtin
$ type clear
clear is /usr/bin/clear
$ type m
m is aliased to 'more'
$ type bc
bc is /usr/bin/bc
$ type if
if is a shell keyword
$ type –path cal
/usr/bin/cal
$ type which
which is aliased to 'type –pqth'
$ type greetings
greetings is a function
greeting()
```

```
    {
            echo "Welcome to my world! ";
    }
```

내부지령과 help지령 내부지령은 쉘에서 일정한 내부원천코드부분을 차지하는 지령이다. 그것들은 디스크상에 있는 2진코드로 번역한 date, cal, finger와 같은 지령들로 쉘에 내장되여 있으며 쓸수 있다. 내부지령들은 쉘에 의해 실행된다. bash는 직결방조체계를 가지고 있기때문에 모든 내부지령에 대한 방조를 볼수 있다. help자체도 내부지령이다. 내부지령들의 정확한 목록은 표 11—28에서 보여 준다.

실례 11-18

1. $ **help help**
   help: help [pattern...]
   Display helpful information about bulit-in commands.if PATTERN is specified,gives detailed help on all commands matching PATTERN,otherwise a list of the built-ins is printed.
2. $ **help pw**
   pwd:pwd
   print the current working directory.

지령행처리의 순서를 변경하기 bash는 지령처리순서를 변경할수 있는 3가지 내부지령들을 제공한다. 그것은 지령,내부지령 그리고 허가이다. 내부지령은 처리순서를 조사하는것으로부터 별명들과 함수들로 분리한다. 내부지령과 실행가능한것들은 오직 경로의 탐색하에서만 찾는다. builtin지령은 다만 내부지령과 무시하는 함수만을 조사하고 실행가능한것들은 해당 경로에서 찾는다. enable내부지령은 내부값들을 대입 혹은 해제한다. 기정값으로는 내부값이 허가로 대입되여 있다. 내부지령비허가는 정확한 경로이름을 규정함이 없이 자기원판에서 찾은 실행할수 있는 지령에 대하여서만 허용한다. 내부지령은 –n추가선택을 사용하면 무효로 된다. 실례로 새로운 쉘프로그람작성자들에게 제일 큰 혼란을 주는것에는 스크립트 test로 이름을 붙였다. 왜냐하면 test는 내부지령이기때문에 쉘은 사용자의 스크립트보다도 오히려 그것을 실행하려고 하기때문이다(내부지령은 보통 임의의 실행할수 있는 프로그람보다 앞서 실행된다.). enable –n test라고 입력하면 내부지령 test는 무시되고 사용자의 스크립트가 우선권을 가지게 된다. 추가선택들이 없으면 내부지령 enable은 모든 내부지령들의 목록을 현시한다. 다음의 모든 내부지령들은 《쉘내부지령들》에서 설명되였다.

실례 11-19

1. $ **enable**
   enable .
   enable :
   enable [
   enable alias
   enable bg

```
enable bind
enable break
enable builtin
enable cd
enable command
enable continue
enable declare
enable dirs
.....
enable read
enable readonly
enable return
enable set
enable shift
enable shopt
.....
enable type
enable typeset
enable ulimit
enable umask
enable unalias
enable unset
enable wait
```

2. **enable –n test**
3. **function cd{builtin cd; echo $PWD; }**

설명

1. 아무런 추가선택도 없이 내부지령 enable을 사용하면 bash쉘내부지령들의 종합적인 목록을 현시한다. 실례에서는 목록의 일부를 보여 준다 .
2. –n추가선택을 사용한 test내부지령은 무시된다. 여기에서는 test라는 스크립트가 실행된다. 조작체계의 지령과 같은 이름을 붙이지 말아야 한다. 만약 다른 쉘에서 같은 스크립트를 실행하려고 하면 내부지령무시기능을 쓸수 없다.
3. 함수를 cd라고 한다. builtin지령이 무한순환으로 되도록 함수 cd를 그대로 이름 달아 함수정의를 cd로 한다.

탈퇴상태 지령 또는 프로그람이 탈퇴한 다음 탈퇴상태를 어미프로쎄스로 되돌린다. 탈퇴상태에는 0으로부터 255까지의 수값이 있다. 조건에 따라 프로그람이 탈퇴할 때 상태값이 0이면 지령실행은 성공적으로 된다는것을 의미한다. 탈퇴상태값이 0이 아니면 지령은 몇가지 방법상 틀렸다는것을 의미한다. 쉘이 지령을 찾지 못하면 탈퇴

상태값은 127이다. 말단으로부터 치명적인 신호가 들어 오면 탈퇴상태값은 128이다. 쉘상태변수 ?는 실행된 마지막지령의 탈퇴상태값을 대입한다. 프로그람이 성공인가 실패인가 하는것은 프로그람을 쓴 프로그람작성자에 의해 확정된다.

실례 11-20

1. $ **grep ellie /etc/passwd**
   ellie:MrHJEFd2YpkJY:501:501::/home/ellie:/bin/bash
2. $ **echo $?**
   0
3. $ **grep nicky /etc/passwd**
4. $ **echo $?**
   1
5. $ **grep ellie / junk**
   grep : /junk: No such file or directory
6. $ **echo $?**
   2
7. $ **grip ellie /etc/passwd**
   bash:grip:command not found
8. $ **echo $?**
   127
9. $ **find / -name core** ^C   *User presses Control-C*
10. $ **echo $?**
    130

설명

1. grep프로그람은 /etc/passwd파일안에서 ellie라는 문자를 검사하며 실행결과는 성공이다. /etc/passwd의 행이 현시된다.
2. ?변수는 grep지령의 탈퇴상태값을 대입한다. 성공상태를 가리키는 0을 현시한다.
3. grep프로그람은 /etc/passwd파일안에서 사용자 nicky를 찾을수 없다.
4. grep프로그람은 패턴을 찾을수 없다. ?변수는 0이 아닌 귀환값을 준다. 탈퇴상태 1은 실패를 나타낸다.
5. grep는 /jank파일을 열수 없기때문에 실패한다. grep오유통보문을 표준오유출력인 현시장치에 보여 준다.
6. grep가 파일을 찾을수 없다면 2라는 탈퇴상태값으로 되돌린다.
7. grep지령을 쉘에서 찾지 못한다.
8. 지령을 찾을수 없기때문에 탈퇴상태를 127로 준다.
9. find지령은 Ctrl-C로 SIGINT신호를 보낼 때 새치기된다. Ctrl-C의 신호값은 2이다.
10. 탈퇴상태값(128+신호의 값)으로 되돌아 온다. 즉 128+2이다.

지령행에서 다중지령들 지령행은 여러가지 지령들로 이루어 질수 있다.

매 지령들은 반두점으로 구분되며 한개 지령행은 행바꾸기에서 끝난다. 탈퇴상태는 지령들의 련결중에서 제일 마지막지령에 해당된다.

실례 11-21

$ ls; pwd; date

설명

지령은 왼쪽에서부터 오른쪽으로 가면서 실행된다. 첫번째 그다음은 두번째 순서대로 행바꾸기가 나타날 때까지 계속하여 처리한다

지령의 그룹화 지령은 또한 그룹화할수 있기때문에 모든 출력은 다른 지령에로 파이프되던가 아니면 파일에로 방향바꾸기될수 있다.

실례 11-22

$ (ls; pwd; date) > outputfile

설명

매 지령의 출력이 outfile파일에 보내여진다. 공백들을 구분하기 위해 괄호가 필요하다.

지령들의 조건실행 조건실행은 2개의 지령문자렬을 특수한 문자 &&과 ll에 의해 구별한다. 이 메타문자들가운데서 오른쪽에 있는 지령은 왼쪽에 있는 지령의 탈퇴조건에 따라 실행될수도 있고 실행되지 않을수도 있다.

실례 11-23

$ cc prgm1.c –o prgm1 && prgm1

설명

첫번째 지령이 성공하면(탈퇴상태 0을 가진다.) &&다음에 있는 지령이 실행된다. 즉 cc프로그람이 prgm.c를 성과적으로 번역할수 있으면 실행프로그람 prgm1이 실행된다.

실례 11-24

$ cc prog.c> & err | | mail bob < err

설명

첫번째 지령이 실패하면(탈퇴상태로 령이 아닌 값을 가진다.) ||기호다음의 지령이 실행된다. 즉 cc프로그람은 prog.c를 번역할수 없으며 오유가 err파일에 써지고 사용자 bob는 err파일을 전자우편으로 보내게 된다.

배경에서의 지령 보통 지령을 실행할 때는 전면에서 실행되며 재촉문은 지령이 실행을 끝마칠 때까지 없어 지지 않는다.

그러나 항상 지령이 끝날 때까지 기다릴수는 없다. 지령행의 끝에 &기호를 쓰면 쉘은 즉시에 쉘재촉문을 되돌려 보내고 배경에서 계속 지령을 실행한다. 사용자는 다른 지령을 기동하기 위해 기다리지 않아도 된다. 배경으로부터 일감의 처리결과는 현시장치에 출력된다.

그러므로 배경에서 지령을 실행하려면 지령의 출력을 파일로 방향바꾸기하거나 다른 장치 실례로 인쇄기 등으로 파이프해야 한다. 변수는 배경에 준 마지막일감의 PID번호를 가진다(배경처리에 대해서는 《일감조종》을 볼것).

실례 11-25

1. $ **man sh | lp&**
2. *[1] 1557*
3. $ **kill –9 $!**

설명

1. man지령(UNIX지령에 대한 안내페지들)의 출력은 현시장치에로 파이프된다. 지령행의 끝에 있는 &기호는 배경에 일감을 넣는다는 표시이다.
2. 현시장치에 2개의 수값이 나타난다. 공백으로 구분된 수값은 배경에 대입된것이 첫번째 일감이라는것을 가리키며 두번째 수값은 PID 아니면 이 일감의 독자적인 처리값이다.
3. 쉘재촉문이 즉시에 현시된다. 배경에서 프로그람이 실행되는 동안 쉘은 전경에서 다른 지령을 대기한다. 변수 !는 배경에 넣어 진 제일 최근일감의 PID를 구한다. 잠간 대기하면 이 일감들은 현시장치에 출력되지 않고 삭제된다.

7. 일감조종

일감조종은 배경 혹은 전경에서 jobs라는 프로그람을 선택적으로 실행할수 있게 하는 bash쉘의 강력한 특징이다. 실행되는 프로그람은 프로쎄스 또는 일감이라고 하며 매 프로쎄스는 PID라고 하는 프로쎄스 ID번호를 가진다. 보통 지령행에 입력한 지령은 전경에서 실행되며 그것을 끝내기 위해 Ctrl-C, Ctrl-\ 를 눌러 신호를 보내지 않으면 프로그람이 끝날 때까지 계속 실행한다. 일감조종을 사용하면 배경에 일감을 보내여 계속 실행할수 있다. 또한 Ctrl-Z에 의해 배경에서 실행하는 일감을 중지시킬수 있고 배경의 일감을 삭제할수 있다. job지령목록을 표 11-3에 보여 준다.

기정으로 일감조종은 이미 대입되여 있다(UNIX의 일부 아래판본들은 이 특성을 가지지 않는다.). 만약 무효로 대입되여 있으면 다음의 지령들가운데서 어느 한가지로 다시 대입할수 있다.

형식

set_m	#.bashrc파일안에서 일감조종을 대입한다.
set_o monition	#파일안에서 일감조종을 대입한다.
Bash_m_I	#대화형 bash를 사용할 때 일감조종을 대입한다.

실례 11-26

1. $ **vi**
   [1]+ Stopped vi
2. $ **sleep 25&**
   [2] 4538
3. $ **jobs**
   [2]+ Running sleep 25&
   [1]- Stopped vi
4. $ **jobs –1**
   [2]+ 4538 Running sleep 25&
   [1]- 4537 Stopped vi
5. $ **jobs %%**
   [2]+ 4538 Running sleep 25&
6. $ **fg %1** *# or* ***kill 4537***
7. $ **jobs –x echo %1**
   4537
8. $ **kill %1** *# or* ***kill 4537***
   [1]+ Stopped vi
   Vim: Caught deadly signal TERM
   Vim: Finished.

   [1]+ Exit 1 vi

설명

1. vi편집기를 실행시킨 다음 vi대화조종을 림시 중지시키기 위해 ^Z(ctrl-Z)를 사용할수 있다. 편집기는 배경에서 림시정지되며 통보 <stopped>가 현시된 다음 즉시에 쉘재촉문이 나타난다.
2. 지령의 끝에 있는 &기호는 배경에서 25라는 인수를 주어 sleep지령을 실행시킨다. [2]는 배경에서 실행되는 두번째 일감이라는것을 의미하며 이 일감의 PID는 4538이다.
3. jobs지령은 배경에서 실행되는 현재의 일감을 현시한다.
4. jobs지령에 –1추가선택을 사용하면 배경에서 실행되는 프로쎄스와 PID번호를 현시한다.
5. %%인수는 일감표안에 넣은 제일 최근의 지령을 현시한다.
6. 퍼센트기호와 일감번호를 사용한 fg지령은 전경안에 번호를 가진 일감을 주자는것이다. 번호가 없으면 fg지령은 제일 최근의 배경일감을 전경으로 옮긴다.

7. –x추가선택은 일감의 PID번호를 출력하는데 사용할수 있다. %1은 첫번째 실례에서 중지된 vi대화조종을 가리킨다.
8. kill지령은 프로쎄스에 TERM신호를 보내고 그것을 삭제한다. vi프로그람이 삭제된다. kill지령에 인수로 일감번호 아니면 PID번호를 줄수 있다.

표 11-3. 일감조종지령

지 령	의 미
Jobs	실행중의 모든 일감들의 목록
^Z(ctrl-Z)	일감을 림시정지한다. 재촉문이 현시장치에 나타난다.
Bg	배경안에서 중지되였던 일감이 실행되기 시작한다.
fg	배경의 일감을 전경으로 옮긴다.
Stop	배경일감을 림시정지한다.
stty tostop	말단에로 출력을 보내려면 배경일감을 림시정지한다.
Kill	규정된 일감에로 삭제신호를 보낸다.
wait[n]	규정된 일감을 대기하고 탈퇴상태값을 되돌려 보낸다.
	여기서 n은 PID 혹은 일감번호이다.

Jobs지령에 대한 인수	표 현
%n	일감번호 n
%문자렬	문자렬로 시작하는 일감이름
%?문자렬	문자렬을 포함하는 일감이름
%%	현재일감
%+	현재일감
%-	현재일감전의 일감
-r	실행되고 있는 모든 일감들을 현시
-s	중지된 모든 일감들을 현시

새로운 jobs추가선택 bash 2.X에서는 jobs지령에 새로운 추가선택들이 첨부되여 있다. 여기에는 –r와 –s추가선택들이 있다. -r추가선택은 진행되고 있는 모든 일감들을 현시하며 –s추가선택은 중지된 모든 일감들을 현시한다.

Disown내부지령 disown내부지령(bash 2. x)은 일감표로부터 특수한 일감들을 삭제한다. 일감이 삭제된 다음 쉘은 쓸모 있는 일감처리는 실행하지 못하고 처리 ID번호에 의해 참고만 할수 있다.

제 2 절. 지령행지름길

1. 지령과 파일이름의 완성

bash는 지령과 파일이름완성을 진행할 때 지령이나 파일이름의 일부분만을 입력하고 타브건을 누르면 나머지부분을 자동적으로 완성한다.

만약 지령행에 첫 문자를 입력하고 타브건을 누르면 bash는 지령이 완성되였다고 판단하고 그것을 실행한다. bash가 파일이름이나 지령이 존재하지 않기때문에 완성할수 없으면 말단에서는 경고음이 울리고 유표는 지령행의 끝에 머물러 있게 된다. 같은 문자들로 시작되는 지령이 더 있는 상태에서 타브건을 누르면 같은 문자로 시작하는 모든 지령들이 현시된다.

실례 11-27

1. $ **ls**
   file1 file2 foo foobrackle fumble
2. $ ls **fu***[tab]* *# Expands to filename to* ***fumble***
3. $ ls **fx***[tab]* *# Terminal beeps,nothing happens*
4. $ ls **fi***[tab]* *# Expands to* ***file_*** *(_ is a cursor)*
5. $ ls **fi***[tab][tab]* *# Lists all possibilities file1 file2*
6. $ ls **foob***[tab]* *# Expands to* ***foobarckle***
7. $ **da***[tab]* *# Completes the* ***date*** *command* *date*
   Tue Feb 28 18:53:40 PST 2001
8. $ **ca***[tab][tab]* *# Lists all commands starting with* ***ca***

   cal *captoinfo case cat*

설명

1. 현재작업등록부의 모든 파일들이 현시된다.
2. fu를 건입력한 다음 타브건을 누르면 fumble이라는 정확한 파일이름이 나타난다.
3. fx로 시작되는 파일이 없기때문에 말단으로부터 경고음이 울리고 유표는 그 자리에 머물러 있으며 아무런 동작도 하지 않는다.
4. fi로 시작되는 파일의 번호이다. 파일이름은 그리 길지 않는 문자들로 완성된다. 만약 또 한번 타브건을 누르면 똑 같은 모든 파일들이 현시된다.
5. 타브건을 두번 누르면 file로 시작되는 모든 파일들이 출력된다.
6. 타브건을 누르면 파일이름은 foobarckle로 확장된다.
7. da라고 입력하고 타브건을 누르면 오직 da로 시작하는 지령은 date지령뿐

이라는것을 알수 있다. 지령이름은 확장되여 집행된다.

8. ca를 입력한 다음 타브건을 누르면 ca로 시작되는 지령이 한개이상이기때문에 다시 타브건을 눌러 ca로 시작되는 모든 지령들을 현시할수 있다.

2. 리력

리력기구는 지령행에 입력한 지령들의 기록번호인 리력목록을 가진다. 등록가입대화조종에서 입력한 지령들은 쉘기억기의 리력목록안에 기억된다. 그다음 기억되여 있는 리력파일을 확인할수 있다. 사용자는 지령을 다시 입력하지 않아도 리력목록으로부터 다시 호출할수도 있고 실행할수도 있다. history내부지령은 리력목록을 현시한다. 리력파일의 기정이름은 .bash_history이며 홈등록부에 있다. bash가 리력파일을 호출할 때 HISTORY변수는 리력파일로부터 리력목록에 얼마나 많은 지령을 복사하여 넣을수 있는가를 규정한다. 기정크기는 500이다. HISTFILE변수는 지령이 기억되는 지령리력파일(~/.bash history가 기정으로 대입되여 있다.)의 이름을 규정한다. 대입할수 없으면 지령리력은 대화형쉘이 탈퇴할 때 기억되지 않는다.

리력파일은 어느 한 등록가입대화조종으로부터 다음것까지 계속 증가한다. HISTFILE SIZE변수는 리력파일안에 넣을수 있는 최대행수를 조종한다. 이 변수에 값을 대입할 때 리력파일의 행수가 초과되면 크기를 조종할수 있다. 기정값은 500이다. fc-1 지령은 리력파일내용을 현시할수도 있고 편성할수도 있다.

표 11-4. 리력변수

FCEDIT	fc지령을 사용하는 UNIX편집기의 경로이름이다.
HISTCMD	현재지령의 리력번호 혹은 리력목록안에서 색인번호이다. HISTCMD가 대입되여 있지 않거나 또는 다시 대입되면 규정된 속성들을 잃어 버리게 된다.
HISTCONTROL	Ignorespace값으로 대입되면 공백문자로 시작되는 행들을 리력목록에 입력한다. 만약 ignoredups값을 대입하면 마지막리력행에 해당되는 행이 입력되지 않는다. Ignoreboth값은 두가지 추가선택을 포함한다. 만약 대입되여 있지 않거나 더 큰 값으로 대입되면 모든 행들은 처리기에 의해 읽어 지고 리력목록에 기억된다.
HISTFILE	지령목록을 기억하는 파일을 규정한다. 기정은 ~/.bash_history이다. 대입되여 있지 않으면 지령리력은 대화형쉘이 탈퇴될 때 기억되지 않는다.
HISTFILESIZE	최대행수가 리력파일안에 규정되여 있다. 이 변수에 값을 주면 리력파일의 크기는 제한된다. 필요하면 행수보다 크지 않은 범위의 값으로 규정한다. 기정값은 500이다.
HISTIGNORE	두점으로 구분된 패턴목록은 지령행들이 리력목록에 기억되는가를 결정할 때 사용된다. 매 패턴은 행의 시작을 고착시키며 문자들로 이루어 진 쉘패턴으로 이루어 진다. 실례로 ty??:&와 같이 ty로 시작하는 지령들중에서 그다음 련이어 2개 물음표로 이루어 지는 지령들을 history지령에서 무시하도록 하는데 &기호를 사용할수 있다.
HISTSIZE	지령리력안에 남아 있는 지령수 기정값은 500이다.

3. 리력파일로부터 지령의 호출

방향건 리력파일로부터 지령을 호출할 때에는 리력파일을 우와 아래로 왼쪽과 오른쪽으로 움직이기 위해 방향건을 사용할수 있다. 사용자는 지우기, 갱신, 공백 등에 표준건을 사용함으로써 임의의 행을 편성할수 있다.

수정한 다음 곧 Enter건을 누르면 행은 다시 편성된다.

표 11-5. 방향건

↑	리력목록의 웃방향으로 이동한다.
↓	리력목록의 아래방향으로 이동한다.
→	history지령의 오른쪽으로 이동한다.
←	history지령의 왼쪽으로 이동한다.

history내부지령 history내부지령은 동등한 수값으로 처리되는 입력한 지령들의 리력을 현시한다.

실례 11-28

1. $ **history**

```
982 ls
983 for i in 1 2 3
984 do
985 echo $i
986 done
987 echo $i
988 man xterm
989 adfasdfasdfasdfasdfasdfasdfasdfasdf
990 id –gn
991 id –un
992 id –u
993 man id
994 more /etc/passwd
995 man ulimit
996 man bash
997 man baswh
998 man bash
999 history
1000 history
```

설명

1. history내부지령은 리력목록으로부터 수값으로 현시된 지령들의 목록을 현시한다. *가 붙은 행들은 변경된다.

fc지령 fc 또는 fix라고 하는 지령은 두가지로 사용될수 있다.

(1) 리력목록으로부터 지령들을 선택하려고 할 때
(2) vi 혹은 emacs편집기중 어느 하나로 지령들을 편성하려고 할 때나 체계우에서 임의의 편집기를 대상으로 할 때

첫번째 경우는 fc에 –l추가선택을 함께 쓰면 리력목록으로부터 어떤 범위의 행들이나 규정된 행들을 선택할수 있다. -l추가선택이 대입되면 출력은 현시장치에로 간다. 실례로 기정으로 대입된 fc –l은 리력목록으로부터 마지막 16개 행들을 현시하며 fc –l 10은 목록의 끝에서부터 10개의 행들을 선택한다. 그리고 fc –l –3은 마지막 3개 행들을 선택한다. -n추가선택은 리력목록안의 지령번호달기를 해제한다. 이 추가선택이 대입되면 지령들의 범위를 선택할수 있고 파일에로 그것을 다시 대입할수 있다.-r추가선택은 지령들의 순서를 재정돈한다. fc의 두번째 형태는《지령행편성》에서 설명한다.

표 11-6. fc지령

fc인수	의 미
-e편집기	편집기안으로 리력목록을 넣는다.
-l n-m	n부터 m까지 범위의 지령들을 현시한다.
-n	리력목록의 번호달기를 해제한다.
-r	리력목록의 순서를 재정돈한다.
-s문자렬	문자렬로 시작하는 지령들을 호출한다.

실례 11-29

1. $ **fc –l**

```
4.  ls
5.  history
6.  exit
7.  history
8.  ls
9.  pwd
10. clear
11. cal 2000
12. history
13. vi file
14. history
15. ls –l
16. date
17. more file
18. echo a b c d
19. cd
20. history
```

2. *$ **fc –1 –3***

   19. cd
   20. history
   21. fc –1

3. $ **fc –ln**

   exit
   history
   ls
   pwd
   clear
   cal 2000
   history
   vi file
   history
   ls –1
   date
   more file
   echo a b c d
   cd
   history
   fc –1
   fc –1 –3

4. $ **fc –ln –3 > saved**

5. $ **more sasved**

   fc –1
   fc –1 –3
   fc –ln

6. $ **fc –1 15**

   15. ls –1
   16. date
   17. more file
   18. echo a b c d
   19. cd
   20. history
   21. fc –1
   22. fc –1 –3
   23. fc –ln
   24. fc –ln –3 > saved
   25. more saved
   26. history

7. $ **fc –1 15 20**
   15. ls –1
   16. date
   17. more file
   18. echo a b c d
   19. cd
   20. history

설명

1. fc 1은 리력목록으로부터 마지막 16개 지령들을 선택한다.
2. fc 1 –3은 리력목록으로부터 마지막 3개 지령들을 선택한다.
3. fc에 ln추가선택을 같이 쓰면 행번호없이 리력목록을 출력한다.
4. 리력목록으로부터 행번호없이 마지막 3개의 지령들을 saved파일에로 출력한다.
5. saved파일의 내용을 현시한다.
6. 15번부터의 지령들을 현시한다.
7. 15번부터 20번까지의 지령들을 현시한다.

fc에 －s추가선택을 주면 문자렬패턴은 이미전의 지령을 재실행할수 있다. 실례로 fc –s rm은 재실행하기 위한 패턴 rm을 포함하는 제일 마지막행이라는것을 나타낸다. Korn쉘의 redo지령을 모방하기 위해서는 r라는 bash별명들을 만들수 있다. 별명 r='fc–s'이므로 지령행에서 r vi라고 입력하면 패턴을 포함하는 리력항목을 재실행한다.이 경우 vi편집기는 어떤 인수를 가지고 제일 마지막으로 기동된다.

실례 11-30

1. $ **history**
   1 ls
   2 pwd
   3 clear
   4 cal 2000
   5 history
   6 ls –1
   7 date
   8 more file
   9 echo a b c d
2. $ **fc –s da**
   date

   Thu Jul 15:33:25 PST 2001
3. $ **alias r="fc –s"**
4. $ **date +%T**

18:12"32

5. **$ r d**
   date +%T
   18:13:19

설명

1. 내부 지령 history는 리력목록을 현시한다.
2. fc에 –s추가선택을 같이 쓰면 da로 시작되는 제일 마지막에 사용한 지령을 검사한다. Date지령을 리력목록안에서 찾게 되며 재실행된다.
3. r라는 별명에 지령 fc –s를 대입한다. 이것은 지령행에 r라고 입력될 때마다 fc –s와 바꾸어 진다.
4. date지령이 실행된다. 현재시간을 현시한다.
5. 별명은 fs –s지령을 지름화한것처럼 사용된다. D로 시작되는 제일 마지막으로 사용한 지령을 재실행한다.

리력지령의 재실행 리력목록으로부터 지령들을 재실행하기 위해 감탄부호(!)를 사용한다. 2개의 !!를 입력하면 리력목록안의 제일 마지막지령이 재실행된다. !다음에 번호값을 입력하면 번호로 지적되는 지령이 재실행된다. !와 문자 혹은 문자렬을 입력하면 문자 또는 문자렬로 시작되는 제일 마지막지령이 재실행된다.

(^)부호 역시 이미전의 지령을 편성하기 위한 지름방법처럼 사용된다.

리력교환문자들의 완성목록을 표 11-7에서 보여 주었다.

표 11-7. 바꿔넣기와 리력

사건지시자	의 미
!	리력교환의 시작을 가리킨다.
!!	이미전의 지령을 재실행한다.
!N	리력목록으로부터 n번째 지령을 재실행한다.
!-N	현재지령으로부터 뒤로 n번째 지령을 재실행한다.
!문자렬	문자렬로 시작하는 제일 마지막지령을 재실행한다.
!?문자렬	문자렬을 포함하는 리력목록으로부터 제일 최근의 지령행인수를 재실행한다.
!$	현재지령행에서 제일 마지막리력지령으로부터 제일 마지막인수를 사용한다.
!!문자렬	이미전의 n번째 지령에 문자렬을 추가하고 실행한다.
!N 문자렬	리력목록안의 N번째 지령에 문자렬을 첨부하고 실행한다.
:!N:S/낡은/새로운	이미전의 n번째 지령에서 새로운 문자렬로 제일 첫번째 출현되는 낡은 문자렬을 바꿔넣기한다.
N:gs/낡은/:!새로운	이미전의 n번째 지령에서 새로운 문자렬로 낡은 문자렬을 전반적으로 바꿔넣기한다.
낡은^새로운^	제일 마지막리력지령에서 새로운 문자렬로 낡은 문자렬을 바꿔넣기한다.

(표계속)

사건지시자	의 미
지령 !N:wn	n번째 지령에 인수(wn)를 첨부하여 현재지령을 실행한다. wn은 이미전의 지령에서 단어의 번호를 지시하는 0, 1, 2,...라는 번호이다. 단어 0은 지령자체를 의미하며 1은 첫번째 인수를 의미한다(실례 11-32에서 보여 준다.).

실례 11-31

1. $ **date**
   Mon Jul 12 12:27:35 PST 2001
2. $ **!!**
   date
   Mon Jul 12 12:29:26 PST 2001
3. $ **!106**
   date
   Mon Jul 12 12:29:26 PST 2001
4. $ **!d**
   date
   Mon Jul 12 12:30:09 PST 2001
5. $ **dare**
   *dare:*Command not found.
6. $ **^r^t**
   date
   Mon Jul 12 12:33:25 PST 2001

설명

1. UNIX지령 date는 지령행에서 실행된다. 리력목록이 갱신된다. 이 지령은 목록에서 제일 마지막지령으로 된다.
2. !!는 리력목록으로부터 제일 마지막지령을 준다. 지령은 재실행된다.
3. 리력목록으로부터 지령번호 106을 재실행한다.
4. 문자 d로 시작하는 리력목록우에서 제일 마지막지령을 재실행한다.
5. 지령이 틀리게 입력되였다. 그것은 date이지 dare가 아니다.
6. ^부호는 리력목록우에서 제일 마지막지령으로부터 문자들의 바꿔넣기에 사용된다. r와 제일 먼저 일치되는것을 at로 바꿔넣기한다. 즉 dare는 date로 된다.

실례 11-32

1. $ **ls file1 file2 file3**
   file1 file2 file3

$ **vi !:1**

vi file1

2. $ **ls file1 file2 file**
   file1 file2 file3
   $ **ls !:2**
   ls file2
   file2
3. $ **ls file1 file2 file3**
   $ **ls !:3**
   ls file3
   file3
4. $ **echo a b c**
   a b c
   $ **echo !$**
   echo c
   c
5. $ **echo a b c**
   a b c
   $ **echo 1^**
   echo a
   a
6. % **echo a b c**
   a b c
   %**echo !***
   echo a b c
   a b c
7. % **!!:p**
   echo a b c

설명

1. ls지령은 file1, file2 그리고 file3을 현시한다. 리력목록은 갱신된다. 지령행은 0이라는 단어값으로 시작하는 단어들로 처리된다. 만약 단어의 앞에 두점이 있으면 리력목록으로부터 반출할수 있다. !:1은 리력목록의 제일 마지막지령으로부터 첫번째 인수를 준다는것을 의미하며 문자렬로 바꿔넣기된다. 마지막지령으로부터 첫번째 인수는 file1이다(단어 0은 지령자체이다.).
2. !:2는 마지막지령의 두번째 인수(file2)로 바꿔넣기하며 ls에 인수를 준다. file2가 현시된다(file2는 세번째 단어이다.).
3. ls!:3을 읽고 리력목록의 제일 마지막지령에서 네번째 단어를 얻으며 인수로

ls지령을 통과한다(file3은 네번째 단어이다.).
4. !감탄부호와 $화폐기호는 리력목록우에서 제일 마지막지령의 제일 마지막인수를 말한다. 마지막인수는 C이다.
5. ^는 지령다음의 첫번째 인수를 의미한다. 감탄부호(!)와 ^기호를 함께 쓰면 리력목록우에서 제일 마지막지령의 첫번째 인수를 말한다. 제일 마지막지령의 첫번째 인수는 a이다.
6. *는 지령다음의 모든 인수들을 의미한다. !와 *는 리력목록우의 제일 마지막지령의 모든 인수들을 의미한다.
7. 리력목록으로부터 제일 마지막지령이 현시되지만 실행되지는 않는다. 리력목록이 갱신된다. 행우에서 ^기호바꿔넣기를 진행할수 있다.

지령행편집 bash쉘은 2개의 내부편집기인 리력목록을 효과적으로 편성할수 있는 emacs와 vi를 제공한다. 지령행에서 편집을 특색 있게 할 때 readline함수는 건들이 정확한 기능들을 수행하도록 한다. 실례로 emacs에서 Ctrl-P는 지령행리력안에서 웃방향으로 현시장치두루말기를 하게 하며 vi에서 k건은 리력목록을 웃방향으로 움직인다. readline은 또한 방향건들, 유표움직임, 변경, 삭제, 본문삽입, 수정, 다시 하거나 하지 않기 등을 조종한다. readline의 다른 특징은《지령과 파일이름의 완성》에서 해설한다. 지령행에서 방조를 주는 readline서고를 보면 더 많은 특징들을 알수 있다. emacs내부편집기는 기정내부편집기이고 본보기이다. vi편집기는 두가지 방식으로 작업하는데 하나는 행우에서 지령을 실행하는것이고 다른 하나는 본문을 입력하는것이다. UNIX에서는 이 편집기들중의 하나를 쓸수 있다. vi편집기를 쓰기 위해서는 다음에 보여 주는 set지령을 첨부[5]하고 ~/.bashrc파일안에 이 행을 넣으면 좋다. vi를 설정하기 위해서는 재촉문이나 혹은 ~/.bashrc파일들중의 어느 하나에 다음과 같이 입력하여야 한다.

실례 11-33

```
set –o vi
```

설명

리력목록의 지령행편성에 내부vi편집기를 설정한다.
Emacs편집기를 설정하기 위해서는 다음과 같이 입력해야 한다.

실례 11-34

```
set –o emacs
```

설명

리력목록의 지령행편집에 내부emacs편집기를 설정한다.

Vi내부편집기 리력목록을 편성하기 위해서는 지령행상태에서 Esc건을 누른다. 그다음 리력목록안에서 웃방향으로 현시장치두루말기를 하기 위해서는 k건을 누른다.

---

5. set -o(편집기)가 설정되지 않았지만 EDITOR 변수가 emacs 나 vi 로 설정되여 있으면 bash 가 정확히 사용한다.

또한 아래로 움직이기 위해서는 J건[6]을 누른다. 편성하려고 하는 지령을 찾을 때에는 왼쪽과 오른쪽을 그리고 삭제, 삽입, 본문변경 등을 위해서는 vi편집기에서 사용하는 표준건들을 사용할수 있다(표 11-8에서 보여 준다.). 편집을 한 다음에는 Enter건을 눌러야 한다. 지령은 리력목록의 항목을 실행하고 첨부하게 된다.

표11-8. vi지령

지 령	기 능
리력파일 움직이기:	
Esc k or +	리력목록을 우로 움직인다.
Esc j or -	리력목록을 아래로 움직인다.
G	리력목록안의 첫번째 행에로 움직인다.
5G	문자렬을 위해 리력목록안의 여섯번째 지령을 움직인다.
/string	리력파일에 대하여 웃방향으로 검색한다.
?	리력파일에 대하여 아래방향으로 검색한다.
행우에서 임의의 곳으로 움직이기:	
h	행우에서 왼쪽으로 움직인다.
I	행우에서 오른쪽으로 움직인다.
b	단어를 뒤방향으로 움직인다.
e or w	단어를 앞방향으로 움직인다.
ˆ or o	행우에서 첫번째 문자의 시작위치에로 움직인다.
$	행의 끝으로 움직인다.
vi에서의 편성 :	
a A	본문을 추가한다.
:I	본문을 삽입한다.
dd dw x	완충기(행, 단어 혹은 문자)에서 본문을 지운다.
cc C	본문을 변경한다.
u U	본래상태
yy Y	완충기안에 행을 복사한다.
p P	행 아래 혹은 행우를 복사하거나 삭제하여 넣는다.
r R	행우에서 본문의 한개 문자 혹은 일부분을 바꿔넣기한다.

emacs내부편집기 emacs내부편집기를 사용하면 vi보다 더 효과적이다. 리력파일을 웃방향으로 움직이기 위해서는 ˆP를 누르고 아래방향으로 움직이기 위해서는 ˆN을 누른

6. vi는 매우 민감하기때문에 큰문자 J와 작은 문자 j는 서로 다른 지령으로 된다.

다. 본문을 변경하거나 수정하기 위해 emacs편집지령들을 사용한 다음에는 꼭 Enter건을 눌러야 지령이 재실행된다. 표 11-9에서 보여 준다.

표 11-9. emacs지령

지 령	의 미
Ctrl-P	리력파일을 우로 움직인다.
Ctrl-N	리력파일을 아래로 움직인다.
Ctrl-B	한개 문자를 뒤방향으로 움직인다.
Ctrl-R	문자렬을 뒤방향으로 검사한다.
ESC B	한개 단어를 뒤방향으로 움직인다.
Ctrl-F	한개 문자를 앞방향으로 움직인다.
ESC F	한개 단어를 앞방향으로 움직인다.
Ctrl-A	단어의 시작위치에로 움직인다.
Ctrl-E	행의 끝으로 옮긴다.
ESC <	리력파일의 첫번째 행에로 움직인다.
ESC >	리력파일의 마지막행에로 움직인다.
emacs에서의 편집:	
Ctrl-U	행을 삭제한다.
Ctrl-Y	공백행을 넣는다.
Ctrl-K	현재유표위치로부터 행의 끝까지 삭제한다.
Ctrl-D	한개 문자를 삭제한다.
ESC D	앞방향의 한개 단어를 삭제한다.
ESC H	뒤방향의 한개 단어를 삭제한다.
ESC공백건	유표위치에 표식을 설정한다.
Ctrl-X ctrl-x	유표과 표식을 호상 바꿔넣기한다.
Ctrl-P ctrl-Y	완충기안에 유표위치로부터 표식까지의 부분을 넣고(ctrl-p) 그것을 그 다음부분에 넣는다(ctrl-Y).

FCDDIT와 편집지령 fc지령에 –e추가선택을 주고 련속하여 리력목록으로부터 선택된 리력지령들을 포함하게 되는 UNIX편집기를 주면 fc –e vi -1 –3은 vi편집기를 요구하며 vi완충기안의 리력목록으로부터 마지막 3개의 지령을 가지고 /tmp안에 림시 파일을 구성한다. 지령은 편성하거나 혹은 설명문을 줄수 있다. 사용자가 편집기를 탈퇴하면 지령들은 모두 현시되고 련이어 실행된다.[7] 편집기이름을 주지 않으면

7. 사용자가 기억하거나 편집기에서 탈퇴하거나 아니면 편집기를 탈퇴하려고 시도하면 즉 지령들을 설명하거나 삭제하지 않는 한 모두 실행된다.

FCEDIT변수의 값을 사용하며(실지로는 초기화파일 bash_profile이나 혹은 .profile 들중의 하나안에 대입되여 있다.) FCEDIT값이 대입되여 있지 않으면 EDITOR변수의 값을 사용한다. 편성을 끝마쳤으면 편집기에서 탈퇴하며 편성된 모든 지령들은 현시되고 련이어 실행된다.

실례 11-35

1. $ **FCEDIT=/bin/vi**
2. $ **pwd**
3. $ **fc**
   <Starts up the full screen vi editor with the pwd command on line 1>

4. $ **history**
   1. date
   2. ls –1
   3. echo "hello"
   4. pwd
5. $ **fc –3 –1** *# Start vi, edit, write/quit, and execute*
   # last 3 commands.

설명

1. FCEDIT변수는 vi, emacs 등과 같은 임의의 UNIX본문편집기의 경로이름을 설정할수 있다. 설정하지 않으면 vi편집기가 기정으로 설정된다.
2. pwb지령은 지령행에 건입력된다. 리력파일안에 들어 간다.
3. fc지령은 편성창문안에 입력한 제일 마지막지령으로 편집기를 대입한다(FCEDIT안에 설정한다.). 사용자가 편집기를 쓰고 벗어 난 다음 입력한 임의의 지령이 실행된다.
4. history지령은 제일 마지막으로 입력한 지령들을 현시한다.
5. fc지령은 편집기의 완충기안에 있는 리력파일로부터 제일 마지막 3개 지령들로 편집기를 쓸수 있게 한다.

4. 별명

별명은 지령에 사용하기 위한 bash사용자정의략어이다. 별명은 만약 지령이 다시 기억하기가 시끄러운 추가선택의 번호나 인수, 문법을 가지고 있을 때 효과적으로 쓰이게

된다. 지령행에서 대입한 별명은 자식쉘들에서는 쓸수 없다. 별명은 보통 .bashrc파일이 새로운 쉘에 기동될 때 실행되는데 이것은 별명들이 새로운 쉘에서 다시 대입되기때문이다. 별명은 또한 쉘스크립트안으로 통과되여야 하지만 스크립트안에 직접 대입되지 않았다면 가능한껏 간단하여야 한다.

별명목록제시 alias내부지령은 설정된 모든 별명들의 목록을 현시한다. 먼저 별명이 현시되고 련이어 지령이나 표현묶음을 현시한다.

실례 11-36

*$ **alias***
alias co='compress'
alias cp='cp –i'
alias mroe='more'
alias mv='mv –i'
alias ls='ls –colorztty'
alias uc='uncompress'

설명

Alias지령은 지령에 해당한 별명을 현시하며 실지지령은 =부호다음에 현시된다.

별명만들기 alias지령은 별명을 만드는데 사용된다. 첫번째 인수는 지령에 붙는 별명의 이름이다. 행은 별명이 실행될 때 사용되는 지령이나 지령묶음으로 구성된다. bash별명은 인수들을 가질수 없다(11장 3절 8.《함수정의》에서 보여 준다.). 다중지령은 반두점으로 분리되며 지령들은 단일인용부호에 의해 인용되는 공백들과 메타문자들을 포함한다.

실례 11-37

1. $ **alias m=more**
2. $ **alias more=more**
3. $ **alias lF= ' ls –alF '**
4. $ **alias r= 'fc –s'**

설명

1. more지령에 m을 설정한다.
2. more지령의 별명을 mroe로 설정한다.
3. 별명정의는 =부호 오른쪽에 공백이 있기때문에 인용부호안에 넣었다. 별명 IF는 지령 ls_aIF로 설정한다.
4. 별명 r는 리력목록으로부터 지령들을 재호출할 때 fc –s대신에 사용된다. 실례로 r vi는 vi문자를 포함하는 리력목록안의 제일 마지막지령을 재실행한다.

별명삭제하기 unalias지령은 별명을 삭제하기 위해 사용된다. 별명을 일시적으로 해제하기 위해서는 \ 기호를 주고 별명이름을 주면 된다.

실례 11-38

1. $ **unalias mroe**
2. $ \ **ls**

설명

1. unalias지령은 정의된 별명목록에서 별명 mroe를 지운다.
2. 별명 ls는 다만 지령실행에서 림시로 해제된다.

5. 등록부탄창조작

사용자가 여러개의 똑같은 등록부들이 있는 등록부나무구조를 등록부탄창안에 넣어 탄창을 조작하므로 cd지령으로 우아래를 검사하면 등록부호출을 쉽게 할수 있다. pushd 내부지령은 탄창으로 등록부들을 넣으며 popd지령은 그것을 없앤다(실례 11-39). 탄창은 탄창안에 넣어 진 제일 최근의 등록부들이 들어 있는 등록부들의 목록이다.등록부들은 내부지령 dirs로 현시할수 있다.

dirs내부지령 내부지령 dir를 –l추가선택과 함께 쓰면 완전한 경로이름형태로 매 등록부들의 등록부탄창을 현시한다. 추가선택을 주지 않으면 dirs는 홈등록부를 현시한다. a+n추가선택을 사용하면 dirs는 0으로 시작되는 등록부목록안의 왼쪽에서부터 계산하여 n번째 등록부입구점을 현시한다. -n추가선택을 사용하면 등록부목록을 오른쪽에서부터 시작하지만 결과는 0과 꼭 같다.

pushd와 popd지령 인수로 등록부를 주면 pushd지령은 등록부탄창에 새로운 등록부를 첨부하는것과 동시에 그 등록부에로 변경한다. 인수로 a+n(여기서 n는 수값이다.)을 주면 pushd는 탄창의 제일 우에서부터 채워 져서 왼쪽에서 시작된다고 볼 때 탄창에서부터 n번째 등록부를 준다. a-n추가선택을 사용하면 같은것을 수행하지만 단지 오른쪽에서부터라는데 차이가 있다.아무런 인수도 없으면 pushd는 등록부탄창의 제일 우에 있는 2개의 원소들을 서로 바꿔넣기한다.

popd지령은 탄창의 제일 우에서부터 등록부들을 없애며 등록부를 변경한다. +n추가선택을 지정(n은 수값)하면 popd는 dirs결과의 왼쪽에서부터 n번째 입구점을 삭제한다.

실례 11-39

1. $ **pwd**

   /home/ellie

   $ **pushed..**

   /home ~

   $ **pwd**

   /home
2. $ **pushed** *#Swap the top directories on the stack*

~/home

$ **pwd**

/home/ellie

3. $ **pushed perlclass**

   ~/perlclass ~ /home

4. $ **dirs**

   ~/perlclass ~/home

5. $ **dirs –1**

   /home/ellie/perlclass/home/ellie   /home

6. $ **popd**

   ~/home

   $ **pwd**

   /home/ellie

7. $ **popd**

   /home

   $ **pwd**

   /home

8. $ **popd**

   bash:popd:Directory stack empty.

설명

1. 첫번째 pwd지령은 한개의 작업등록부 /home/ellie를 현시한다. 다음인수로 ..을 주면 pushd는 어미등록부를 등록부탄창에 넣는다. pushd의 출력은 등록부환경의 제일 우인 /home을 가리킨다(현시된 목록의 왼쪽에서부터 시작한다.). 그리고 사용자의 홈등록부는 ~문자에 의해 재현시된다. pushd는 또한 탄창안에 넣어 진것으로 등록부를 변경한다. 새로운 등록부는 두번째 pwd지령에 의해 현시된다.
2. 인수가 없는 pushd는 탄창의 제일 우에 있는 2개 등록부들을 서로 바꿔넣기한다. 실례로 등록부는 사용자의 홈등록부 즉 /home/ellie에로 절환된다.
3. pushd의 지령은 인수 ~/perlclass를 탄창안에 넣으며 등록부를 변경한다.
4. 내부지령은 목록의 왼쪽에서 시작하는 제일 웃준위의 등록부탄창을 현시한다. ~는 사용자의 홈등록부를 확장한다.
5. -l추가선택을 주면 dirs 목록은 ~확장기호대신에 정확한 경로이름으로 등록부탄창을 현시한다.

6. popd지령은 탄창의 제일 우에서부터 한개 등록부를 삭제한다. 그다음 등록부를 변경한다. popd지령은 탄차의 제일 우에서부터 다른 등록부를 삭제한 다음 등록부를 변경한다.
7. popd지령은 탄창이 비였기때문에 아무런 등록부입구점도 지울수 없다.

6. 메타문자(통용기호)

메타문자들은 특수한 목적에 사용하는 문자들이다. 쉘메타문자들을 통용기호(wildcard)라고 한다.

표 11-10. 메타문자

메타문자	의 미
\	다음물음표를 해석한다.
&	배경에서 처리된다.
;	지령들을 구분한다.
$	변수들을 바꿔넣기한다.
[abc]	한개 문자에 대응된다.
?	문자들의 묶음가운데서 한개 문자에 대응된다.
	례를 들면 a, b 혹은 c
[!abc]	문자들의 묶음가운데 한개 문자에 대응되지 않는다..
	례를 들면 a, b 혹은 c가 아니다.
*	빈값 또는 그외의 물음표들에 대응된다.
(cmds)	자식쉘에서 지령들을 실행한다.
{cmds}	현재쉘에서 지령들을 실행한다.

7. 파일이름바꿔넣기

지령행은 처리할 때 문자들의 모임으로 이루어 지는 파일이름이나 경로이름들을 지름화하기 위해 메타문자를 사용한다. 파일이름바꿔넣기메타문자들은 알파베타순서로 확장한 표 11-11에서 보여 준다. 메타문자를 파일이름으로 확장하는 처리를 파일이름바꿔넣기 혹은 메타문자파일이름확장(globbing)이라고도 한다. 메타문자를 사용하여 그것과 일치하는 파일이름이 없으면 쉘은 해당한 문자로 메타문자를 처리한다.

표 11-11. 쉘메타문자들과 파일이름바꿔넣기

메타문자	의 미
*	령 혹은 그외의 물음표들에 해당된다.
?	한개 물음표에 대응된다.
[abc]	묶음 a, b 혹은 c에서 한개 물음표에 대응된다.

(표계속)

메타문자	의 미
[!abc]	묶음 a, b 혹은 c가 아닌 한개 물음표에 대응된다.
[a,ile,ax]	문자들의 묶음 혹은 한개 물음표에 대응된다.
[!a-z]	a부터 z까지 범위밖의 한개 문자에 해당한다.
\	메타문자에서 벗어 나거나 혹은 무효로 된다.

별표(*) 별표(*)는 파일이름에서 빈값 혹은 한개이상의 문자들에 해당되는 메타문자이다.

실례 11-40

1. $ **ls ***
   abc abc1 abc122 abc123 abc2 file1 file1.bak file2.bak none
   nonsense nobody nothing nowhere one
2. $ **ls *.bak**
   file1.bak file2.bak
3. $ **echo a***
   ab abc1 abc122 abc123 abc2

설명

1. *는 한개 작업등록부안에서 모든 파일들을 의미한다. 모든 파일들은 ls에서 인수로 사용되며 현시된다.
2. 빈값 또는 하나이상의 물음표들로 시작하며 .bak로 끝나는 모든 파일들에 해당되고 현시된다.
3. a로 시작되며 령 또는 그이상 물음표들로 이루어 지는 모든 파일들에 해당되며 echo지령에 인수로 사용되였다.

물음표(?) 물음표는 파일이름에서 한개 물음표를 나타낸다. 파일이름이 한개 혹은 그이상의 물음표로 이루어 질 때 쉘은 파일이름에 맞는 문자로 물음표를 바꿔넣어 실행한다.

실례 11-41

1. $ **ls**
   abc1 abc122 abc2 file1.bak file2.bak nonsense nothing one
   abc1 abc123 file1 file2 none noone nowhere
2. $ **ls a?c?**
   abc1 abc2
3. $ **ls ??**
   ls: ??: No such file or directory

4. $ **echo abc???**
   abc122 abc123

5. $ **echo ??**
   ??

설명

1. 현재등록부에 있는 파일들이 현시된다.
2. a로 시작되고 련이어 한개 문자가 있고 다음 c와 a문자로 이루어 진 파일이름들을 현시한다.
3. 2개 물음표로 이루어 지는 파일들을 현시하는데 파일을 찾지 못했기때문에 오유통보문이 현시된다.
4. abc로 시작되고 련이어 3개문자로 이루어 지는 파일이름이 echo지령에 의해 현시된다.
5. 2개 문자들로 이루어 지는 파일이 없다. 이와 같은 파일을 찾을수 없으므로 그대로 출력한다.

중괄호([ ]) 중괄호는 문자들의 범위나 혹은 묶음속에서 한개 문자를 포함하는 파일이름을 대상할 때 사용된다.

실례 11-42

1. $ **ls**
   abc abc122 abc2 file1.bak nonsense nothing
   one abc1 abc123 file1 file2 none noone nowhere

2. $ **ls abc[123]**
   abc1 abc2

3. $ **ls abc[1-3]**
   abc1 abc2

4. $ **ls [a-z][a-z][a-z]**
   abc one

5. $ **ls [!f-z]???**
   abc1 abc2

6. $ **ls abc12[23]**
   abc122 abc123

설명

1. 현재작업등록부에 있는 모든 파일들을 현시한다.
2. 4개 물음표로 이루어 지는 모든 파일이름들을 대상하는데 여기서는 파일이름이 abc로 시작하고 련이어 중괄호안에 있는것들중 어느 한 문자 즉 1, 2 혹은 3과 대응되는것만을 현시한다.

3. 4개 물음표로 이루어 진 파일이름을 대상하여 현시한다. 파일이름이 abc로 시작되고 련이어 1부터 3까지의 어떤 수값으로 이루어 지는것만을 고찰한다.
4. 파일이름이 3개문자로 이루어 지는데 모두 소문자알파베타로 이루어 진 파일만을 대상하여 현시한다.
5. 파일이름이 4개 물음표로 이루어 졌는데 첫 문자는 f로부터 z사이가 아닌 문자이고 련이어 어떤 3개 문자로 이루어 지는 파일만을 현시한다.
6. 파일이름이 abc 그리고 련이어 2 아니면 3으로 이루어 진것만을 현시한다.

대괄호확장({ }) 대괄호는 반점(,)으로 구분된 문자렬목록중에서 임의의 한개에 대응된다. 보통 문자렬들은 파일이름들을 쓴다. 여기서 {, }가 있는데 {는 임의의 물음표들을 준비하기 위해 열기할 때 사용하며 }는 문자들을 첨가한 다음 닫으려고 할 때 사용된다. 이것들은 모두 선택적이다. 대괄호안에서 공백들은 인용괄호를 하지 않아도 된다.

실례 11-43

1. $ **ls**
   a.c.b.c abc ab3 ab4 ab5 file1 file2 file3 file4 file5 foo faa fumble
2. $ **ls f{oo,aa,umble}**
   foo faa fumble
3. $ **ls a{.c,c,b[3-5]}**
   a.c ab3 ab4 ab5
4. $ **mkdir /usr/local/src/bash/{old,new,dist,bugs}**
5. $ **chown root /usr/{ucb/{ex,edit},lib/{ex?.*,how_ex}}**
6. $ **echo fo{o,um}***
   *fo{o,um}**
7. $ **echo {mam pap,ba}a**
   mama papa baa
8. $ **echo post{script,office,ure}**
   postscript postoffice posture

설명

1. 현재등록부의 모든 파일들이 현시된다.
2. f로 시작하고 그다음에 문자렬 oo, aa 혹은 umble로 이루어 진 파일들이 현시된다. 대괄호 {} 안에 있는 공백은 대괄호 } 기호를 놓친 오유통보문을 막기 위해 사용한다.
3. 문자 a로 시작하여 다음에 .c, c, 혹은 b3, b4, b5로 이루어 진 파일들이 현시된다. 중괄호 [ ] 는 대괄호 {} 기호안에서 사용할수 있다.
4. \ usr/local/src/bash안에 새로운 4개의 등록부 즉 old, new, dist 그리

고 bugs가 만들어 진다.
5. /usr/ucb등록부의 ex와 exit와 edit라는 파일 그리고 ex다음에 한개 문자, 점, 최소한 한개 문자로 이름 붙은 파일 그리고 \ usr/lib등록부에서 how_ ex라는 파일을 root로 준다.
6. 괄호안에 그 어떤 공백도 인용되지 않았으면 괄호확장을 하지 않는다.
7. 괄호확장은 항상 파일이름확장을 하기 위해 필요한것이 아니다. 이 실례에서 맨 마지막기호 a는 대괄호 {} 안의 매 문자렬을 첨부한다는것이며 확장다음에 다시 반복한다.
8. 맨 마지막은 문자렬 post다음에 괄호안에 있는 반점으로 분리된 목록이다. 괄호확장이 완성되고 결과문자렬이 현시된다.

메타문자벗어나기 내부문자로 메타문자를 사용하려면 메타문자들이 해석되지 못하게 거꿀빗선을 사용해야 한다.

실례 11-44

1. $ **ls**
   abc filel youx

2. $ **echo How are you?**
   How are youx

3. $ **echo How** are **you\ ?**
   How are you?

4. $ **echo When does this line **
   > ever end\ ?
   When does this line ever end?

설명

1. 현재작업등록부안의 모든 파일들을 현시한다.
2. 쉘은 물음표(?)로 파일이름확장을 실현한다. y-o-u로 시작되며 련이어 한개 물음표로 이루어 지는 모든 파일들이 대상되며 바꿔넣기된다. 파일이름 youx는 How are youx를 읽기 위해 문자렬을 바꿔넣기한다.
3. \ 프로쎄스에 의해 물음표(?)처리를 뛰여 넘었으며 쉘이 그것을 해석하지 않았다는것을 의미한다.
4. 행바꾸기는 \ 처리에 의해 뛰여 넘는다. 2차재촉문이 나타날 때까지 현시된다. 물음표(?)는 파일이름확장으로부터 보호하기 위해 뛰여 넘는다.

~와 ~~확장 ~문자는 경로이름확장에서 bash쉘(C쉘로부터)에 의해 접수된다. ~자체는 사용자의 홈등록부에 대한 정확한 경로이름을 준다.[8] ~에 사용자이름을 첨부하면 그 사용자의 완전한 경로이름을 준다. ~다음에 +를 첨부하면 PWD(현재의 작업등록부) 값을 준다. OLDPWD 역시 이미전의 작업등록부에 해당된다.

---

8. 이중인용부호 혹은 단일인용부호안에 있지 않으면 ~문자는 확장되지 않는다.

실례 11-45

1. $ **echo ~**

   /home/jody/ellie

2. $ **echo ~joe**

   /home/joe

3. $ **echo ~ +**

   /home/jody/ellie/per1

4. $ **echo ~ -**

   /home/jody/ellie/prac

5. $ **echo $OLDPWD**

   /home/jody/ellie/prac

6. $ **cd –**

   /home/jody/ellie/prac

설명

1. ~는 홈등록부의 정확한 경로이름을 준다.
2. 사용자이름앞에 ~를 주면 정확한 joe의 홈등록부경로이름을 준다.
3. ~+표기는 작업등록부의 완전경로이름을 표시한다.
4. ~-표기는 이전의 작업등록부의 완전경로이름을 표시한다.
5. OLDPWD변수는 이전의 작업등록부를 포함한다.
6. -는 이전의 작업등록부를 가리킨다. 이전의 작업등록부에로 등록부를 변경하여 그 등록부를 현시한다.

통용기호조종하기 bash noglob변수를 대입하거나 혹은 set지령에 –f추가선택을 주면 파일이름바꿔넣기는 무효로 된다. 이것들은 보통 grep, sed 혹은 awk프로그람들에서 패턴검사에 사용된다. 만약 통용기호가 설정되지 않았으면 모든 메타문자들은 통용기호해석을 해제하기 위해 \ 기호로부터 벗어 나야 한다. shopt내부지령(bash판본 2.x)은 메타문자파일이름확장을 조종하는 추가선택을 유지한다.

실례 11-46

1. $ **set noglob** *or* **set –f**
2. $ **print * ??[]-$LOGNAME**

   ** ?? [] /home/jody/ellie ellie*

3. $ **unset noglob** *or* **set +f**

4. $ **shopt –s dotglob**   *#Only available in bash versions 2.x*

5. $ **echo *bash ***
   .bash_history .bash_logout .bash_profile .bashrc bashnote
   bashtest

설명

1. –f추가선택을 set지령에 인수로 준다. 파일이름확장을 위해 사용되는 *의 의미가 무효로 된다.
2. 파일이름확장메타문자는 해석함이 없이 그대로 현시된다. ~와 $는 파일이름확장에 사용되지 않는다.
3. noglob를 해제하든지 +f추가선택을 대입하든지 둘중의 하나에 의해 파일이름메타문자는 확장된다.
4. shopt내부지령은 쉘에 추가선택을 대입하도록 한다. dotglob추가선택은 모든 메타문자들을 쓸수 있도록 허용한다.
5. 4행에서 dotglob추가선택이 설정되였다. 그러므로 파일이름확장에 *를 사용할 때 파일이름이 패턴 bash를 포함하고 있으면 점(.)으로 시작되는 모든 파일이름들을 확장한다.

확장된 파일이름확장(bash 2.x) 패턴대응에 대한것들을 Korn쉘에서 고찰하면 bash 2.x는 규칙적인 표현문법으로 사용하는 확장기능들을 포함한다. 규칙적인 표현연산자는 shopt지령을 대입하는 extglob추가선택이 설정되여 있지 않으면 인식할수 없다.

shopt –s extglob

표 11-12. 확장된 패턴대응하기

규칙표현	의 미
abc?(2\ 9)|	물음표(?)는 괄호안의 임의의 패턴중에서 빈값 혹은 한개 문자를 일치시킨다. 수직막대기기호는 <or>조건을 의미한다. 실례로 2나 9가 일치되여 abc21, abc91 혹은 abc1을 나타낸다.
abc*([0-9])	*는 괄호안에 임의의 패턴중에서 빈값 아니면 여러개가 있음을 의미한다. abc다음에 빈값 혹은 다른 수자들 실례로 abc, abc1234, abc3, abc2 등을 나타낸다.
abc+([0-9])	+는 괄호안에 임의의 패턴중에서 한개 아니면 여러개가 있음을 의미한다. abc 다음에 한개 혹은 그이상의 수자들 실례로 abc3, abc123 등을 나타낸다.
no@(one|ne)	@는 괄호안에 있는 임의의 패턴중에서 무조건 한개가 있음을 의미한다.
no!(thing| where)	!는 괄호안에 있는 임의의 패턴을 제외한 모든 문자만을 의미한다(즉 no, nobody 혹은 noone). 하지만 nothing이나 nowhere는 아니다.

실례 11-47

1. $ **shopt –s extglob**
2. $ **ls**
   abc abc122 f1 f3 nonsense nothing one
   abc1 abc2 f2 none noone nowhere
3. $ ls **abc?(1|2)**
   abc abc1 abc2
4. $ ls **abc*([1-5])**
   abc abc1 abc122 abc2
5. $ **ls abc+([0-5])**
   abc1 abc122 abc2
6. $ ls **no@(thing|ne)**
   none nothing
7. $ ls **no!(thing)**
   none nonsense noone nowhere

설명

1. shopt내부지령에 extglob(extended globbing)추가선택을 대입하여 사용하면 bash는 문자들에 대응되는 확장된 패턴들을 인식할수 있다.
2. 현재작업등록부의 모든 파일들이 현시된다.
3. abc로 시작되고 련이어 괄호안의 패턴들중 어느 하나로 빈값이 아니면 한개에 의해 형성되는 파일이름들이 대응된다. abc, abc1아니면 abc2이다.
4. abc로 시작되고 련이어 1과 5사이에서 빈값이 아니면 그이상 수값들로 형성되는 파일이름이 대응된다. abc, abc1, abc122, abc123 그리고 abc2이다.
5. abc로 시작되고 련이어 0으로부터 5사이의 한개 혹은 여러개의 수값들로 형성되는 파일이름이 대응된다. abc1, abc122, abc123 그리고 abc2이다.
6. no로 시작되고 련이어 틀림없이 무엇이 있든가 혹은 ne로 형성되는 파일이름이 대응된다. nothing 혹은 none이다.
7. no로 시작되고 련이어 틀림없이 무엇이 있는 파일이름이 대응된다. none, nonsense, noone 그리고 nowhere이다. !는 아니라는것을 의미한다.

제 3 절. 변수

변수의 형 변수의 형에는 2가지 즉 국부변수와 환경변수가 있다. 국부변수는 오직 그것을 만든 쉘에서만 쓸수 있다. 환경변수는 그것을 만든 쉘에서 처리되는 모든 자식프로쎄스들에서 효력이 있다. 몇개의 변수들은 사용자에 의해 만들어 진것이며 다른것들은 특수한 쉘변수들이다.

이름짓기약속 변수이름은 알파베타 혹은 밑선문자로 시작해야 한다. 쓸수 있는 변수들은 알파베타, 10진수(0으로부터 9) 그리고 밑선문자들이다. 그외의 물음표들은 변수이름의 끝을 현시하는데 사용한다. 변수에 값을 줄 때에는 같기부호(=)안에 공백을 포함시키지 말아야 한다. 변수에 값을 대입하기 위해서는 같기부호(=)다음에 행바꾸기를 주어야 한다. 국부변수를 만들기 위해서는 다음의 형식으로 값을 주어야 한다.

형식

```
변수=값
```

실례 11-48

```
name=Tommy
```

declare내부지령 2개의 내부지령 declare와 typeset지령은 변수설정을 조종하는 추가선택들을 사용하여 변수들을 만드는데 사용된다. typeset지령(Korn쉘로부터)은 실지로는 declare(bash)지령과 같다. bash방조문서에는 《typeset지령은 Korn쉘에서 제공되며 declare내부지령과 약간의 차이를 가지고 있다.》[9]고 지적되여 있다. 인수들을 사용하지 않으면 declare는 대입된 모든 변수들을 현시한다. 보통 읽기만 하는 변수들은 다시 대입될수 없거나 대입할수도 없다. 옹근수형변수들은 declare로 대입될수 있다.

형식

```
변수=값
```

실례 11-49

```
declare name=Tommy
```

표 11-13. declare추가선택

추가선택	의 미
-f	함수이름과 정의들을 현시한다.
-r	읽기만 하는 변수들을 현시한다.
-x	자식쉘에 변수이름을 넘긴다.
-i	옹근수형변수들을 현시한다.
-a[ㄱ]	배렬변수를 대입한다. 즉 원소들을 대입한다.
-F[ㄱ]	함수이름을 현시한다.

ㄱ. **-a** 그리고 F는 bash 2.X에서만 쓰인다.

1. 국부변수와 유효범위

변수의 범위는 프로그람안에서 효력을 가지는 장소를 말한다. 쉘에서 국부변수들의

[9] bash 참고서 : http://www.delorie.com/gnu/docs/bash/bashref_56.html.

범위는 변수를 만든 쉘에서만 유효하다. 값을 대입할 때 변수라고 하면 갈기부호다음에 공백은 될수 없다. 변수에 빈값을 대입하기 위해서는 =부호다음에 행바꾸기가 있어야 한다.[10]

$기호는 변수에 기억된 값을 뽑아 내기 위해 변수의 앞에 대입한다. local함수는 국부변수를 만들기 위해 사용할수 있는데 함수내에서만 사용된다(《함수정의하기》에서 보여 준다.).

국부변수의 대입 국부변수는 변수이름에 간단히 값을 주거나 혹은 실례 11 - 50에서 보여 준 declare내부함수를 사용하여 대입할수 있다.

실례 11-50

1. $ **round=world** *or* **declare round=world**
   $ **echo $round**
   world
2. $ **name="Peter piper"**
   $ **echo $name**
   Peter Piper
3. $ **x=**
   $ **echo $x**
4. $ **file.bak="$HOME/junk"**
   bash:file.bak=/home/jody/ellie/junk:not found

설명

1. 국부변수 round에는 world라는 값이 대입된다. 쉘이 변수이름앞에서 $(화폐기호)를 만나면 변수바꿔넣기를 진행한다. 변수의 값이 현시된다(변수바꿔넣기에 사용되는 $와 함께 사용한 재촉문 $는 혼돈되지 않는다.).
2. 국부변수 name에는 값 "Peter Piper"가 대입된다. 인용부호는 공백처리를 위해 필요된다. 그러므로 쉘은 지령행을 해석판단할 때 단어구별에서 문자렬을 분리하지 않는다. 변수의 값이 현시된다.
3. 국부변수 x에는 값을 대입하지 않는다. 거기에는 빈값이 대입된다. 빈값, 빈문자렬이 현시된다.
4. 변수이름이 맞지 않는다. 변수이름에는 오직 수자, 문자 그리고 밑선(_)만을 쓸수 있다. 쉘은 문자렬을 지령으로 처리한다.

실례 11-51

1. $ **echo $$**
   1313
2. $ **round=world**
   $ **echo $round**
   world

---

10. 값 혹은 빈값으로 설정된 변수는 set 지령에 의해 표시되지만 해제변수는 그렇게 되지 않는다.

```
3.  $ bash              # Start a subshell
4.  $ echo $$
    1326
5.  $ echo $round
6.  $ exit              # Exits this shell, returns to parent shell
7.  $ echo $$
    1313
8.  $ echo $round
    world
```

설명

1. 2중화페기호변수($$)의 값은 현재쉘의 PID를 준다. 쉘의 PID는 1313이다.
2. 국부변수 round에는 문자렬 world가 대입된다. 그리고 이 변수의 값을 현시한다.
3. 새로운 bash쉘이 기동된다. 이것을 보조쉘 또는 자식쉘이라고 한다.
4. 이 쉘의 PID는 1326이다. 어미쉘의 PID는 1313이다.
5. 국부변수 round는 지령쉘에서 정의되지 않는다. 빈행이 현시된다.
6. exit지령은 쉘을 탈퇴하며 어미쉘에로 되돌아 간다(Ctrl-D도 이 쉘을 탈퇴시킨다.).
7. 어미쉘에로 되돌아 간다. 그의 PID가 현시된다.
8. 변수 round의 값이 현시된다. 이 쉘에서 국부변수이다.

읽기전용변수의 설정 읽기전용변수는 재정의할수 없거나 해제할수 없는 특수한 변수이다. declare함수를 사용하면 읽기전용변수가 재정의되지만 해제될수 없다.

실례 11-52

```
1.  $ name=Tom
2.  $ readonly name
    $ echo $name
    Tom
3.  $ unset name
    bash: unset: name: cannot unset: readonly variable
4.  $ name=Joe
    bash: name: readonly variable
5.  $ declare –r city='Santa Clara'
6.  $ unset city
    bash:unset:city:cannot unset:readonly variable
7.  $ declare city='San Francisco'     # what happened here?
    $ echo $city
    San Francisco
```

설명

1. 국부변수 name에 값 Tom을 대입한다.
2. 변수는 읽기전용으로 만들어 진다.
3. 읽기전용변수는 해제할수 없다.
4. 읽기전용변수는 재정의할수 없다.
5. declare내부지령은 읽기전용변수 city에 값 Santa clara를 대입한다. 인용부호는 공백을 포함하는 문자렬을 대입할 때 필요하다.
6. 변수는 읽기전용변수로 대입한 다음 해제할수 없다.
7. 읽기전용변수는 declare지령으로 만들어 질 때 해제할수 없지만 다시 대입된다.

2. 환경변수

환경변수는 그것을 만든 쉘과 그와 바꿔넣기된 자식쉘이나 혹은 프로쎄스에서 효력이 있다. 그것들은 국부변수와 구별하기 위해 전역변수라고도 한다. 약속에 따라 환경변수는 가장 중요한 변수이다. 환경변수들은 export내부지령에 의해 반출할수 있는 변수들이다. 변수를 만든 쉘을 어미쉘이라고 한다. 만약 새로운 쉘이 어미쉘로부터 기동되였다면 이것을 자식쉘이라고 부른다.

환경변수는 자기를 만든 쉘로부터 기동한 임의의 자식프로쎄스에서 효력이 있다. 그것들은 어미로부터 자식에로, 자식으로부터 손자에로 이어 진다. 하지만 다른 방향으로는 통과하지 못한다. 즉 자식프로쎄스는 환경변수를 만들수 있지만 어미프로쎄스에로 이어지지 못하고 오직 자식프로쎄스에서만 효력을 가진다.[11]

몇개의 환경변수들 실례를 들어 HOME, LOGNAME, PATH 그리고 쉘은 /bin/login프로그람에서 등록가입전에 대입되여야 한다. 보통 환경변수들은 사용자의 홈등록부의 bash_쉘안에 정의되고 기억된다. 환경변수목록을 표 11 - 15에 보여 준다.

환경변수들의 설정 환경변수를 설정하기 위한 export지령은 값을 대입한 다음에나 혹은 변수를 대입할 때 사용된다. declare내부지령에 –x추가선택을 주면 같은 동작을 한다(그것을 반출할 때 변수에 $를 사용하지 않는다.).

형식

```
export 변수=값
변수=값;export 변수
declare –x 변수=값
```

실례 11-53

```
export NAME=john
PS1= '\ d: \ W:$USER> ' ; export PS1
declare –x TERM=sun
```

11. DNA처럼 계승은 어미로부터 자식으로 즉 한 방향으로만 간다.

표 11 -14. Export지령과 추가선택

추가선택	값
- -	추가선택프로세스의 끝을 현시한다. 존재하는 파라메터들은 인수들이다.
-f	이름 - 값쌍이 함수로 취급된다. 변수가 아니다.
-n	국부변수를 일반변수로 바꾼다. 변수는 자식프로세스로부터 반출하지 않는다.
-p	모든 일반변수들을 현시한다.

실례 11-54

1. $ **export TERM=sun** *# or* **declare –x TERM=sun**
2. $ **NAME="John Smith"**
   $ **export NAME**
   $ **echo $NAME**
   John Smith
3. $ **echo $$**
   319 *# pid number for parent shell*
4. $ **bash** *# Start a subshell*
5. $ **echo $$**
   340. # pid number for new shell
6. $ **echo $NAME**
   John Smith
7. $ **declare –x NAME="April Jenner"**
   $ **echo $NAME**
   April Jenner
8. $ **exit**
   #Exit the subshell and go back to parent shell
9. $ **echo $$**
   319. # pid number for parent shell
10. $ **echo $NAME**

John Smith

설명

1. TERM변수에 sun을 대입한다. 변수가 동시에 반출된다. 현재쉘로부터 기동된 프로쎄스가 변수를 넘겨 받는다. Declare –x를 사용할수도 있다.
2. 변수 NAME은 쉘로부터 기동된 자식쉘들에서도 쓸수 있도록 반출된다.
3. 이 쉘의 PID이름이 현시된다.
4. 새로운 쉘이 기동된다. 새로운 쉘을 자식이라고 하며 원래의 쉘은 어미라고 한다.
5. 새로운 bash 쉘의 PID는 $$변수에 기억되며 그의 값이 현시된다.
6. 어미프로쎄스안에서 대입된 변수는 새로운 쉘에서 반출되고 현시된다.
7. 내부 declare함수는 변수를 대입하기 위해 다른 방법을 사용한다. –x추가선택을 붙여 declare를 사용하면 반출하기 위한 변수를 알려 준다. 변수는 April Jenner로 다시 대입된다. 모든 자식쉘에로 반출되지만 어미프로쎄스에서는 효력이 없다. 반출된 값들은 자식쉘에로 이어 지지 않는다.
8. bash자식쉘이 탈퇴한다.
9. 어미처리의 PID가 다시 현시된다.
10. 변수 NAME은 원래의 값을 가진다. 변수들은 어미로부터 자식쉘에로 반출될 때 값들을 되돌려 준다. 자식은 어미의 변수값을 변경시킬수 없다.

표 11-15. bash환경변수

변수이름	의 미
_(밑선)	이전 지령을 마지막인수로 한다.
BASH	bash를 실행하기 위해 사용하는 정확한 경로이름을 준다.
BASH_ENV	ENV와 같지만 bash2.0 혹은 그이상에서만 대입한다.ㄱ
BASH_VERSINFO	판본 2.0 혹은 그이상에서 bash의 판본에 대한 통보를 준다.ㄱ
BASH_VERSION	bash의 판본을 준다.
CDPATH	cd지령을 위한 탐색경로. 이것은 쉘이 cd지령에 의해 규정된 목적등록부들을 찾는 옹근두점으로 분리된 등록부목록이다.
COLUMNS	설정되면 쉘편집방식에 대한 편집창문의 너비와 선택지령을 정의한다.
DIRSTACK	bash판본이 2.0이상일 때 등록부탄창의 현재내용이다.ㄱ
EDITOR	내부편집기의 경로이름이다. 실례로 emacs, gmacs 혹은 vi 등이다.
ENV	스크립트를 포함하여 새로운 bash쉘이 기동될 때 실행되는 환경파일이다. 보통 이 변수에 대입된 파일이름은 .bashrc이다. ENV의 값은 경로이름으로 해석되기전의 파라메터표현식, 지령바꿔넣기, 산수연산표현식으로 된다.
FUID	쉘기동으로 초기화된 현재사용자의 ID를 효과적으로 확장한다.
FCEDIT	fc지령을 위한 기정편집기이름.
FIGNORE	파일이름완성을 진행할 때 무시하는 두점으로 분리된 뒤붙이들의 목록이다. FIGNORE안의 입구점들중 하나에 맞먹는 파일이름을 파일이름에 해당되는 목록으로부터 실행된다. 실례값은 0이다.

(표계속)

변수이름	의 미
FORMAT	지령파이프행에서 time의 출력을 형식화하기 위해 사용된다.
GLOBIGNORE	파일이름을 확장하는 동안 무시되는 파일들의 목록.ㄱ
GROUPS	현재사용자가 소속되는 그룹의 배렬.ㄱ
HISTCMD	현재지령의 리력목록안에서 리력번호 혹은 색인. HISTCMD가 대입되지 않으면 규정된 속성들을 잃어 버린다. 그다음 다시 대입하여도 결과는 같다.
HISTCONTROL	ignorespace의 값을 대입하면 공백문자로 시작하는 행들은 리력목록에 입력되지 않는다. Ignoredups의 값을 대입하면 마지막리력행에 대응되는 행들은 입력되지 않는다. ignoreboth의 값은 2개의 추가선택을 조합한다. 대입하지 않거나 그이상의 값을 대입하면 해석프로그람에 의해 읽어 진 모든 행들은 리력목록우에 기억된다.
HISTFILE	지령리력을 기억한 규정된 파일. 기정으로 ~/.bash_history이다. 대입하지 않으면 지령리력은 대화형쉘이 끝날 때 기억되지 않는다.
HISTFILESIZE	리력파일에 포함되는 최대행수. 이 변수가 값을 가질 때 리력파일은 제한을 받는다. 필요하면 행번호보다 더 크지 않게 규정할 수 있다. 기정으로 500이다.
HISTSIZE	지령들의 번호가 지령리력안에 들어 간다. 기정은 500이다.
HOME	규정된 등록부가 없을 때 cd지령에서 사용하는 홈등록부.
HOSTFILE	쉘이 기본이름을 완성하려고 할 때 읽게 되는 /etc/hosts와 같은 형식으로 파일이름을 포함한다. 파일이 대화형에서 변화되면 기본이름완성을 기다리게 되는데 bash는 이미 존재하는 자료기지에로 새로운 파일의 내용을 첨부한다.
HOSTTYPE	bash가 실행되고 있는 기계의 형태를 자동적으로 대입한다. 기정은 체계에 관련된다.
IFS	내부마당분리기호들인 공백건, 타브, 행바꾸기문자는 지령바꿔넣기, 순환구조에서의 목록, 입력읽기로부터 마당들을 단어들로 분리하는데 리용한다.
IGNOREEOF	단독입력으로 EOF문자를 접수할 때 쉘의 동작을 조종한다. 설정되면 값은 bash가 끝나기전에 입력행의 첫 문자들로 입력된 련속적인 EOF문자들의 수이다. 값은 존재하지만 수값이 아니거나 값이 없을 때에는 기정값으로 10이다. 존재하지 않으면 EOF는 쉘에 입력의 끝을 알린다. 이것은 대화형쉘들에서만 효력이 있다.
INPUTRC	기정으로 대입된 ~./inputrc를 무시하고 readline기동파일이름을 준다.
LANG	LC_로 시작하는 변수가 특별히 대입되여 있지 않는 국부목록을 확장하기 위해 사용한다.ㄱ
LC_ALL	LANG값을 제외한 다른 모든 LC_변수.ㄱ
LC_COLLATE	경로이름의 확장과 표현식의 범위, 똑 같은 클라스들을 구별할 때와 그리고 경로이름들과 패턴들을 대응시킬 때 대조순서를 끝낸다.ㄱ

(표계속)

변수이름	의 미
LC_MESSAGES	$로 처리되는 2중인용부호안의 물음표를 현시하기 위해 사용하는 국부변수를 확장한다.ㄱ
LINENO	항상 파라메터를 참고하며 쉘은 함수나 혹은 스크립트안에 현재의 행번호(1로 시작된다.)를 현시하는 10진수를 바꿔넣기한다.
MACHTYPE	Bash가 실행되고 있는 체계를 서술하는 문자렬을 포함한다.ㄱ
MAIL	이 파라메터가 우편파일의 이름으로 설정되고 MAILPATH파라메터가 설정되지 않으면 쉘은 사용자에게 지정된 파일에 우편이 도착했다는것을 알려 준다.
MAIL_WARNING	대입되였으면 우편을 검사하는 파일을 호출한 다음 검사하며 통보문 《우편을 기억할 파일이름을 읽었다. 》를 현시한다.
MAILCHECK	이 파라메터는 흔히 MAILPATH 혹은 MAIL파라메터에 의해 규정된 파일에서 얼마만한 시간(s)에 쉘이 전자우편도착을 검사하는가를 규정한다. 기정값은 600s이다. 령으로 설정하면 매개 1차재촉문이 나타나기전에 쉘이 검사한다.
MAILPATH	두점으로 구별된 파일목록이다. 이 파라메터를 대입하면 쉘은 임의의 규정된 파일안에 우편도달자를 통보한다. 매 파일이름다음에는 %와 변경시간을 변화시킬 때 현시하는 통보문을 놓을수 있다. 기정통보문은 《사용자는 전자우편을 가진다.》이다.
OLDPWD	제일 마지막작업등록부
OPTRAG	제일 마지막추가선택인수값은 getopts내부지령에 의해 처리된다.
OPTERR	1로 대입하면 getopts내부지령으로부터 오유통보문을 현시한다.
OPTIND	getopts내부지령에 의해 처리되게 하기 위한 다음인수의 색인
OSTYPE	자동적으로 bash를 실행하는 조작체계를 가리키는 문자렬을 대입한다.
PATH	지령들의 검색경로이다. 쉘이 지령들을 조사할 때의 두점으로 구별된 등록부목록이다. 기정경로는 체계에 관련되며 bash를 설치하는 관리자에 의해 설정된다. usr/gnu/bin:/usr/local/bin:/usr/ucb: /bin:/usr/bin:이다.
PIPESTATUS	파이프형의 제일 마지막으로 실행된 전경일감들에 해당되는 상태값목록을 포함하는 묶음
PPID	어미프로쎄스의 ID를 처리한다.
PROMPT_COMMAND	이미전의 재촉문을 현시하기전에 실행되는 이 변수를 지령에 대입한다.
PS1	1차재촉문문자렬이다. 기정으로 $이다.
PS2	2차재촉문문자렬이다. 기정으로 >기호이다.
PS3	select지령에서 사용되는 선택재촉문문자렬이다. 기정으로 #?이다.
PS4	추적이 실행될 때 사용되는 오유수정재촉문문자렬이다. 기정으로 +이고 추적은 set_x로 동작할수 있다.
PWD	현재작업등록부 cd로 대입한다.
RANDOM	이 파라메터는 항상 참고한다. 우연수를 발생한다. 우연수의 순서는 RANDOM을 대입하지 않으면 주어 진 속성을 가지지 않게 되며 다시 대입결과는 같다.

(표계속)

변수이름	의 미
REPLY	read인수들이 대입되지 않았을 때 대입한다.
SECONDS	항상 SECONDS는 쉘이 되돌아 온 다음의 초수를 참고한다. SECONDS를 대입하지 않으면 규정된 속성들을 잃게 되며 다시 대입한 다음에 결과는 같다. 만약 값에 SECONDS를 대입하면 허용된 값을 더한 다음 초(s)를 참고하고 귀환한다.
SHELL	쉘은 기동할 때 이 이름으로 환경을 조사한다. 쉘은 기정값으로 PATH, PS1, PS2, MAILCHECK 그리고 IFS, HOME 그리고 MAIL을 주며 login(1)로 대입된다.
SHELLOPTS	사용할수 있는 추가선택목록을 포함한다. 실례로 braceexpand, ha쉘, monitor 등이다.
SHLVL	bash가 기동할 때마다 증가한다.
TMOUT	탈퇴전의 입력대기회수를 규정한다.
UID	쉘기동시에 초기화되는 현재사용자 ID를 확장한다.

ㄱ. bash 판본 2.x 에서는 사용할수 없다.

변수설정해제 국부변수와 환경변수들을 대입하지 않는 한 unset지령을 사용하여 해제할수 있다.

실례 11-55

```
unset name;unset TERM
```

설명

unset지령은 쉘로부터 변수들을 지운다.

변수값인쇄: echo지령 내부지령 echo는 표준출력에로 인수들을 인쇄한다. Echo지령에 –e추가선택을 같이 사용하면 출력표현을 조종할수 있는 ESC코드를 사용할수 있다. 표 11-16에서 echo추가선택과 ESC코드를 보여 준다 .

표 11-16. echo추가선택과 ESC코드

추가선택	의 미
-e	ESC코드의 해석을 허용한다.
-n	출력형의 끝에 행바꾸기를 준다.
-E[ㄱ]	ESC문자들의 해석을 무효로 한다(bash 2.X).
ESC코드	
\ a[ㄱ]	경고음
\ b	공백건
\ c	행바꾸기없이 행만을 현시한다.
\ f	용지제공

(표계속)

추가선택	의 미
\ n	행바꾸기
\ r	되돌이
\ t	타브
\ v	수직타브
\ \	\ 기호
\ nnn	ASCII코드 nnn에 해당한 문자(8진수)

ㄱ. bash 판본 2.x 에서는 사용할수 없다.

ESC코드를 사용할 때 –e추가선택을 사용해야 한다는것을 명심해야 한다.

실례 11-56

1. $ **echo The username is $LOGNAME.**
   The username is ellie.
2. $ **echo –e "\ t\ tHello there\ c"**
   Hello there$
3. $ **echo –n "Hello there"**
   Hello there$

설명

1. echo지령은 현시장치에 인수를 현시한다. 변수바꿔넣기는 echo지령을 실행하기전에 쉘에 의해 실행된다.
2. echo지령에 –e추가선택을 사용하면 c프로그람작성언어에서처럼 ESC코드를 제공한다. $는 쉘재촉문이다.
3. –n추가선택을 대입하면 행은 행바꾸기없이 현시한다. ESC코드는 echo의 판본에 의해 제공되지 않는다.

printf지령 printf[12]는 출력형식현시를 위해 사용할수 있다. C언어의 printf함수와 꼭같이 형식화한 문자렬을 현시한다. 형식화는 출력을 어떤 형식으로 할것인가를 나타내는 문자렬로 이루어 진다. 형식화구조는 %다음에 메타문자(dioux XfeEgGcs)로 설계된다. 여기서 %f는 류동소수점, %d는 10진수를 나타낸다.

지령행재촉문에 printf를 – –help와 함께 입력하면 printf를 어떻게 사용하고 완성하는가를 방조 받을수 있다. Printf의 판본을 보기 위해서는 printf –-–version을 입력해야 한다. bash 2.x를 사용하면 내부지령printf는 /usr/bin에서 진행되는 판본과 같은 형식을 사용한다.

12. bash 판본 2.x 에서 printf 는 내부지령이다.

형식

printf 형식 [인수...]

실례 11-57

Printf "%10.2f%5d\ n" 10.5   25

표 11-17.   Printf지령을 위한 형식규정자

형식규정	값
\ "	2중인용부호
\ ONNN	세자리수로 대입한 8진문자
\ \	\
\ a	경고음
\ b	공백건
\ c	출력 없다.
\ f	용지제공
\ n	행바꾸기
\ r	행바꾸기건
\ t	수평타브
\ v	수직타브
\ xNNN	NNN 3자리의 16진 문자
%%	단일 %
%b	\ ESC코드처럼 해석되는 인수

실례 11-58

1. $ **printf --version**
   printf (GNU sh-utils) 1.16
2. $ **type printf**
   printf is a shell builtin
3. $ **printf "The number is %.2f\ n" 100**
   The number is 100.00
4. $ **printf "%-20s%-15s%10.2f\ n" "Jody" "Savage" 28**
   Jody   Savage   28.00
5. $ **printf "|%-20s|%-15s|%10.2f|\ n" "Jody" "Savage" 28**

Jody |Savage | 28.00|

6. **$ printf "%s's average was %.1f%%. \ n" "Jody" $ (((80+70+90)/3))**

   Jody's average was 80.0%.

설명

1. printf지령의 GNU판본을 현시한다.
2. bash 2.x를 사용한다면 printf는 내부지령이다.
3. 인수 100은 형식화문자렬에서 규정한 %.2f에 의해 10진수의 오른쪽 자리수를 2자리 택한 류동소수점으로 현시된다. C와는 달리 인수들을 구별하는데 반점이 없다.
4. 형식화문자렬은 3개의 조건을 규정한다. 첫번째에는 %-20s(왼쪽 줄맞추는 20문자 문자렬), 다음은 %-15s(왼쪽 줄맞추기하는 15문자 문자렬) 그리고 마지막으로 %10.2f(오른쪽 줄맞추기 10자리중 소수점아래 2자리인 류동소수점 형식이다. 매 인수는 %부호에 의해 형식화된다. 그러므로 문자렬 jody에는 첫번째 %, 문자렬 savage는 두번째 %, 수값 28은 마지막 %부호에 응답한다.
5. 이 행은 4행에서 수직기호가 첨부되였을뿐 똑같다.
6. printf지령은 문자렬 Jody를 형식화하고 산수연산표현식의 결과를 형식화한다. 2개의 퍼센트기호(%%)는 한개의 퍼센트기호(%)기호를 현시하기 위해 필요하다.

변수확장변경자 변수는 특수한 변경자에 의해 검사되고 변경될수 있다. 변경자는 변수가 대입되여 있으면 검사를 위한 지름조건검사를 제공한다. 그다음 검사에 기초하여 변수에 값을 줄수 있다. 변수확장변경자에 대한 목록을 표 11-18에 보여 준다 .

표 11-18. 변수변경자

변경자	값
${변수:-단어}	변수가 대입되여 있고 빈값이 아니라면 값을 바꿔넣기한다. 그러므로 단어를 바꿔넣기한다.
${변수:=단어}	변수가 대입되여 있거나 혹은 빈값이 아니라면 값을 바꿔넣기한다. 그러므로 단어를 대입한다. 변수의 값이 바꿔넣기된다. 위치파라메터는 이 방법으로 대입하지 말아야 한다.
${변수:+단어}	변수가 대입되여 있고 빈값이 아니라면 단어를 바꿔넣기한다. 아무것과도 바꿔넣기된다.
${변수:?단어}	변수가 대입되여 있고 빈값이 아니라면 값을 바꿔넣기한다. 그러므로 단어를 현시하고 쉘로부터 탈퇴한다. 단어가 생략되면 통보문 <파라메터가 공백이거나 혹은 대입되여 있지 않다.>가 현시된다.
${변수:편차}	변수문자렬에서 편차된 부분문자렬을 준다. 편차의 값은 문자렬의 끝을 0으로 보고 평가한다.ㄱ
${변수:편차:길이}	편차에 해당한 부분문자렬을 준다. 문자렬의 길이가 포함된다.

ㄱ. bash 판본 2.0 에서는 사용할수 없다.

두점과 함께 변경자(-, =, +, ?)를 같이 사용하면 변수가 대입되였는가 아니면 빈값

인가를 검사한다. 두점이 없으면 빈값으로 대입된 변수는 대입되였다고 본다.

실례 11-59

(Substitue Temporary Default Values)

```
1. $ fruit=peach
2. $ echo ${fruit:-plum}
   peach
3. $ echo ${newfruit:-apple}
   apple
4. $ echo $newfruit
5. $ echo $EDITOR          #More realistic example
6. $ echo ${EDITOR:-bin/vi}
   /bin/vi
7. $ echo $EDITOR
8. $ name=
   $ echo ${name-Joe}
9. $ echo ${name-Joe}
   Joe
```

설명

1. 변수 fruit에 값 peach를 대입한다.
2. 특수한 변경자는 변수 fruit가 대입되였으면 검사한다. 만약 대입되여 있으면 현시하고 없으면 fruit는 plum으로 바꿔넣기되고 그 값이 현시된다.
3. 변수 newfruit는 대입되여 있지 않다. 값 apple이 림시로 newfruit에 바꿔넣기된다.
4. 대입은 다만 림시적이다. 변수 newfruit는 대입되지 않는다.
5. 환경변수 EDITOR는 대입되지 않는다.
6. :- 변경자는 /bin/vi로 EDITOR를 바꿔넣기한다.
7. EDITOR도 대입되여 있지 않다. 아무것도 현시하지 않는다.
8. 변수 nome은 빈값으로 대입된다. 두점변경자를 미리 주지 않으면 변수는 대입으로 고찰된다. 또한 빈값으로 되여 있어도 변수 name에 새값 Joe가 대입된다.
9. 두점은 변수를 대입하지 않거나 빈값으로 대입하는것들중 어느 하나를 검사하기 위해 변경자를 조작한다. 이 경우 변수 name에 값 Joe가 바꿔넣기된다.

실례 11-60

(Substituete Permanent Default Values)

```
1. $ name=
```

2. $ **echo ${name:=Peter}**
   Peter
3. $ **echo $name**
   Peter
4. $ **echo ${EDITOR:=/bin/vi}**
   /bin/vi
5. $ **echo $EDITOR**
   /bin/vi

설명

1. 변수 name에 빈값이 대입된다.
2. 특수한 변경자 :=는 변수이름이 대입되였는가를 검사한다. 대입되여 있으면 변경시키지 않으며 빈값이거나 대입되여 있지 않으면 =부호오른쪽의 값을 준다. peter는 변수가 빈값으로 대입된 다음 name에 대입된다. 대입은 변하지 않는다.
3. 변수 name은 여전히 값 peter로 되여 있다.
4. 변수 EDITOR의 값은 /bin/vi로 대입된다.
5. 변수 EDITOR의 값을 현시한다.

실례 11-61

(Substitute Temporary Alternate Value)

1. $ **foo=grapes**
2. $ **echo $(foo:+pears)**

   pears
3. $ **echo $foo**

   grapes

   $

설명

1. 변수 foo는 값 grapes를 가진다.
2. 특수한 변경자 :+는 변수가 대입되여 있으면 검사한다. 만약 대입되여 있으면 림시로 foo를 pears로 바꿔넣기한다. 없으면 빈값이 들어 간다.
3. 변수 foo는 현재 원래의 값을 가진다.

실례 11-62

(Creating Error Message Based On Default Value)

1. $ **echo ${namex:? "namex is undefined"}**
   namex:namex is undefind

2. $ **echo $(y?)**
   y: paremter null or not set

설명

1. :? 변경자는 변수가 대입되여 있으면 검사한다. 대입되여 있지 않으면 ?의 오른쪽 문자렬이 표준오유에서 변수이름다음에 현시된다. 스크립트안에 있으면 스크립트는 탈퇴한다.
2. 통보문이 ? 다음에 제공되여 있지 않으면 쉘은 표준오유에 기정통보문을 현시한다.

실례 11-63

(Creating Substring ㄱ)

1. $ **var=notebook**
2. $ **echo ${var:0:4}**
   note
3. $ **echo ${var:4:4}**
   book
4. $ **echo ${var:0:2}**
   no

설명

1. 변수에 값 notebook가 대입된다.
3. var의 부분문자렬은 편차 0으로 시작되고 notebook에서 n, e에서 끝나는 4개 문자렬의 길이를 가진다.
3. var의 부분문자렬은 편차 4로 시작하므로 notebook에서 b, k에서 끝나는 4개 문자렬의 길이를 가진다.
4. var의 부분문자렬은 편차 0으로 시작하므로 notebook에서 n, o에서 끝나는 2개 문자렬의 길이를 가진다.

부분문자렬의 변수확장 패턴에 대응되는 인수들은 문자렬의 시작 혹은 끝으로부터 문자렬의 일부를 골라 잡기 위해 사용한다. 대다수 지령들은 경로의 앞 혹은 뒤부분으로부터 경로이름부분을 선택하는데 이 연산자를 사용한다.

표 11-19. 변수확장부분문자렬ㄱ

확 장	기 능
${변수 %패턴}	패턴을 변수값중에서 제일 작게 선택되는 부분으로 보고 그것을 지운다.
${변수 %%패턴}	패턴을 변수값중에서 제일 길게 선택되는 부분으로 보고 그것을 지운다.

ㄱ. Bash 판본 2.x 에서는 사용할수 없다.

(표계속)

확 장	기 능
${변수 #패턴}	패턴을 변수값중에서 제일 작게 선택되는 부분으로 보고 그것을 지운다.
${변수 ##패턴}	패턴을 변수값중에서 제일 길게 선택되는 부분으로 보고 그것을 지운다.
${#변수}	변수에서 문자들의 수를 바꿔넣기한다. * 혹은 @이면 길이는 위치파라메터수이다.

실례 11-64

1. $ **pathname="/usr/bin/local/bin"**
2. $ **echo ${pathname%/bin*}**

   /usr/bin/local

설명

1. 국부변수 pathname에 /usr/bin/local/bin을 대입한다.
2. %는 패턴 /bin다음에 빈값 혹은 다른 문자들을 포함하는 pathname가운데서 제일 작게 선택되는 부분을 지운다. 즉 / bin을 해제한다.

실례 11-65

1. $ **pathname="usr/bin/local/bin"**
2. $ **echo ${pathname%%/bin*}**

   /usr

설명

1. 국부변수 pathname에 /usr/bin/local/bin을 대입한다.
2. %%는 패턴 /bin다음에 련이어 빈값 혹은 다른 문자들을 포함하는 pathname가운데서 제일 길게 선택되는 부분을 삭제한다.즉 /bin/local/bin을 해제한다.

실례 11-66

1. $ **pathname=/home/lilliput/jake/.bashrc**
2. $ **echo ${pathname#/home}**

   /lilliput/jake/.bashrc

설명

1. 국부변수 pathname에 /home/liliput/jake/.bashrc를 대입한다.

2. #는 패턴 /home을 포함하는 pathname가운데서 제일 작게 인도되는 부분을 지운다. 즉 경로변수의 시작에 있는 /home이 해제한다.

실례 11-67

1. $ **pathname=/home/lilliput/jake/.bashrc**
2. $ **echo ${pathname##*/}**

   .bashrc

설명

1. 국부변수 pathname에 /home/liliput/jake/.bashrc를 대입한다.
2. ##는 빈값 혹은 다른 문자들이 있으면서 마지막이 /문자로 끝나는 pathname가운데서 제일 길게 선택되는 부분을 삭제한다. 즉 경로이름으로부터 /home/liliput/jake를 해제한다.

실례 11-68

1. $ **name="Ebenezer Scrooge"**
2. $ **echo ${#name}**

   16

설명

1. 변수 name에 문자렬 Ebenezer Scrooge를 대입한다.
2. $(#변수)문법은 변수 name으로 지정된 문자렬의 수를 현시한다. Ebenezer Scrooge는 16개 문자로 구성된다.

위치파라메터 특수한 내부변수들을 위치파라메터라고 하는데 이것들은 지령행으로부터 인수들이 통과할 때 쉘스크립트에서 사용되기도 하고 함수를 통과하는 임의의 값을 보호하기 위해서도 사용된다.

변수는 파라메터목록에서 위치를 나타내는 수자 1, 2, 3 등으로 나타내기때문에 위치파라메터라고 한다. 표 11-20에서 보여 준다.

쉘스크립트의 이름은 변수 $0에 기억된다. 위치파라메터들은 대입할수 있고 set지령으로 해제할수도 있다.

표 11-20. 위치파라메터

표형식	기 능
$0	현재쉘스크립트의 이름을 참고한다.
$1-$9	위치파라메터 1-9
${10}	위치파라메터 10
$#	위치파라메터의 값을 평가한다.
$*	모든 위치파라메터를 평가한다.

(표계속)

표형식	기 능
$@	$*와 같다. 2중인용부호를 쓰면 제외된다.
"$*"	"$1, $2, $3" 등을 평가한다.
"$@"	"$1", "$2", "$3" 등을 평가한다.

실례 11-69

```
1.  $ set punky tommy bert jody
     $ echo $*                # Prints all the positional parameters
     punky tommy bert jody

2.  $ echo $1                 # Prints the first position
     punky

3.  $ echo $2    $3           # Prints the second and third position
     tommy bert

4.  $ echo $#                 # Prints the total number of positional
     4                          # parameters

5.  $ set a b c d e f g h i j k l m
     $ print $10               # Prints the first positional parameter
     a0                         # followed by a 0.
     $ echo ${10} ${11}      # Prints the 10th and 11th positions
     j k

6.  $ echo $#
     13

7.  $ echo $*
     a b c d e f g h I j k l m

8.  $ set file1 file2 file3
     $ echo \ $$#
     $3

9.  $ eval echo \ $$#
     file3

10.  $ set  - -                 # Unsets all positional parameters
```

설명

1. set지령은 위치파라메터들에 값을 대입한다. $특수변수는 대입된 모든 파라

메터모임을 포함한다.
2. 첫번째 위치파라메터의 값 punky를 현시한다.
3. 두번째, 세번째 파라메터의 값 tommy와 bert를 현시한다.
4. $#특수변수는 현재위치파라메터모임의 수를 포함한다.
5. set지령은 모든 위치파라메터들을 다시 대입한다. 원래의 파라메터목록이 지워 진다. 90이상의 임의의 위치파라메터들을 현시하기 위해서는 2개의 수자들을 다같이 { }로 처리해 주어야 한다.
6. 위치파라메터의 번호는 현재 13이다.
7. 모든 위치파라메터들의 값이 현시된다.
8. $기호가 해제되며 $#는 인수들의 번호이다. echo지령은 화폐기호다음에 위치파라메터의 번호가 붙은 $3을 현시한다.
9. eval지령을 실행하기전에 지령행을 해석한다. 우선 지령해석기에 의해 해석되고 $3에 현시되며 eval지령을 준 다음에는 $3의 값과 file3을 현시한다.
10. set지령에 –추가선택을 주면 모든 위치변수들을 지우거나 해제한다.

다른 특수한 변수 쉘은 한개 문자로 구성되는 특수한 번호를 가진다. 문자앞의 $기호는 변수에 기억된 값을 호출하게 한다. 표 11-21에서 보여 준다.

표 11-21. 특수한 변수

변 수	의 미
$	쉘의 PID
-	현재 설정되여야 하는 sh추가선택
?	마지막으로 실행된 탈퇴값
!	마지막일감의 PID가 배경에 넣어 진다.

실례 11-70

1. $ **echo The pid of this 쉘 is $$**
   The pid of this shell is 4725
2. $ **echo The options for this shell are $-**
   The options for this shell are imh
3. $ **grep dodo \ etc\ passwd**
   $ **echo $?**
   1
4. $ **sleep 25&**
   4736
   $ **echo $!**
   4736

설명

1. $변수는 이 프로세스의 PID값을 유지한다.
2. 변수는 대화형bash쉘의 모든 추가선택들을 현시한다.
3. grep지령은 /etc/passwd파일에서 문자렬 dodo를 검색한다. 변수는 제일 마지막으로 실행된 지령의 탈퇴상태를 보존한다. 그다음 grep의 결과가 1로 귀환된다. grep는 검색이 실패한다는것을 알린다. 탈퇴상태 0은 성공적인 탈퇴라는것을 가리킨다.
4. 변수 !는 배경에 위치한 마지막지령의 PID값을 유지한다. 배경에 지령을 보내기 위해 sleep지령에 &가 첨부되였다.

3. 인용

인용은 규정된 메타문자들을 해석되지 않게 하거나 파라메터확장을 막기 위해 사용한다. 인용에는 3가지가 있는데 그것은 \ 기호, 단일 그리고 2중인용부호이다. 표 11-22에 현시한 문자들은 쉘에 의해 규정되고 인용된다.

표 11-22. 인용에 필요한 메타문자

메타문자	의 미
;	지령구문
&	배경처리
()	지령묶음; 자식쉘을 작성한다.
{}	지령묶음; 자식쉘은 만들지 않는다.
|	파이프
<	입력방향바꾸기
>	출력방향바꾸기
행바꾸기	지령완료
공백/타브	단어경계부호
$	변수바꿔넣기문자
*[]?	파일이름확장을 위한 쉘메타문자

단일인용과 2중인용은 대응되여야 한다. 단일인용은 $, *, ?, |, > 그리고 <와 같은 규정된 메타문자들을 해석되지 않게 보호한다.

2중인용부호 역시 메타문자들을 해석되지 않게 보호하지만 처리하기 위한 변수와 지령바꿔넣기문자($기호와 ′ ′ 기호)들을 허용한다.

단일인용은 2중인용을 보호하며 2중인용은 단일인용을 보호한다.

bourne쉘과는 달리 bash는 사용자가 인용을 놓쳐 버렸다면 통보하여 준다. 대화형쉘이 동작할 때는 인용이 맞지 않을 때 두번째 재촉문이 나타난다.

쉘스크립트안에서는 파일을 검사하여 인용부호의 짝이 맞지 않으면 필요한 인용부호

짝을 통보해 준다. 쉘이 다음인용부호에 대하여 처리할수 없으면 프로그람이 중지되여 말단에 <` "`이 없기때문에 bash는 EOF를 기대할수 없다.>라는 통보를 현시한다.

인용은 쉘프로그람작성자들이 제일 많이 사용한것이다. 지령해석기의 인용규칙을 부록 3에서 보여 준다.

거꿀빗선(\ ) 거꿀빗선(\ )은 한개 문자를 인용하기 위해 사용된다.

\ 이 한개만 있을 때에는 해석되지 않는다. \ 기호는 $기호와 ' '기호들을 보호하며 또한 2중인용부호안에 있으면 해석되지 않는다.

실례 11-71

1. $ **echo Where are you going\ ?**
   Where are you going?
2. $ **echo Start on this line and**
   > go to the next line.
   Start on this line and go to the next line.
3. $ **echo \ **
   \
4. $ **echo '\ \ '**
   \ \
5. $ **echo '\ $5.00'**
   \ *$5.00*
6. $ **echo "\ $5.00"**
   $5.00
7. $ **echo 'Don\ 't you need $5.00?'**
   >
   >'
   Don\ t you need .00?

설명

1. \ 은 ? 표식에 해당한 파일이름바꿔넣기를 하지 못하도록 한다.
2. \ 은 행바꾸기를 제외시켜 다음행이 이 행의 일부분으로 되게 한다.
3. \ 자체는 특수한 문자이기때문에 해석하지 않는다.
4. \ 이 단일인용으로 닫겨 져 있을 때에는 해석하지 않는다.\
5. 단일인용안에 있는 모든 문자는 그대로 처리된다. \ 은 그 어떤 설명도 하지 않는다.
6. 2중인용부호안에 있을 때 \ 은 변수바꿔넣기를 위한 해석으로부터 화페기호를 방지한다.
7. \ 이 단일인용안에 있을 때에는 해석되지 않는다. 그러므로 쉘은 3개의 단일인용부호를 고찰하게 된다(문자렬의 끝에 있는 한개는 대응되지 않는

다.). 2차재촉문이 현시되고 단일인용을 닫을 때까지 대기한다. 쉘이 닫기 인용부호를 줄 때 모든 인용들이 벗겨 지며 echo지령에 의해 문자렬이 현시된다. 우선 2개의 인용부호가 대응하기때문에 문자렬의 나머지부분 <t you need $5.00?>는 그 어떤 인용부호안에도 넣어 지지 않게 되였다. 쉘은 $5의 값을 평가한다. 그것은 빈값이기때문에 .00이 현시된다.

단일인용부호 단일인용부호는 서로 대응되여야 한다. 그것들은 모든 메타문자들이 해석되지 않게 보호한다. 단일인용을 현시하기 위해서는 2중인용부호안에 넣거나 거꿀빗선으로 벗어 나야 한다.

실례 11-72

1. $ **echo 'hi there**
   \> **how are you?**
   \> **When will this end?**
   \> **When the quote is matched**
   \> **oh '**

   hi there
   how are you?
   When will this end?
   When the quote is matched
   oh

2. $ **echo Don\ 't you need '$5.00?'**
   Don't you need $5.00?

3. $ **echo 'Mother yelled, "Time to eat ! "'**
   Mother yeelled, "time to eat ! "

설명

1. 단일인용부호는 행우에서 대응되지 않는다. Bourne쉘은 2차재촉문을 현시한다. 인용부호가 대응될 때까지 대기한다.
2. 단일인용부호는 모든 메타문자들을 해석하지 않게 보호한다. Don't의 '(웃반점은) \ 에 의해 벗어 난다. 따라서 앞에 있는 단일인용부호가 대응되고 그 다음 문자렬의 끝에 있는 단일인용부호가 짝을 맺는다. $와 ?는 단일인용부호쌍안에 들어 있으므로 쉘해석으로부터 보호된다. 즉 문자 그대로 정확히 취급한다.
3. 단일인용부호는 문자렬안에서 2중인용부호를 보호한다.

2중인용부호 2중인용부호들은 변수와 지령바꿔넣기를 할수 있게 대응되여야 하며 쉘해석으로부터 임의의 다른 메타문자들을 보호하여야 한다.

실례 11-73

1. $ **name=Jody**

2. $ **echo "Hi $name, I'm glad to meet you ! "**
   Hi Jody, I'm glad to meet you !

3. $ **echo "Hey $name, the time is $(date)"**
   Hey Jody, the time is Wed Jul 14 14:04:11 PST 2001

설명

1. 변수 name에 문자렬 Jody를 대입한다.
2. 문자렬을 둘러 싸고 있는 2중인용부호들은 재촉문 name에서 $를 제외한 모든 특수한 메타문자들을 해석하지 않게 보호한다. 변수바꿔넣기는 2중인용부호안에서 진행된다.
3. 변수바꿔넣기와 지령바꿔넣기는 둘다 2중인용부호로 둘러 싸여 있을 때 실행된다. 변수 name이 확장되고 괄호안에 있는 date지령이 실행된다.

4. 지령바꿔넣기

지령바꿔넣기는 변수안에 있는 지령의 출력을 대입할 때와 문자렬안에 지령의 출력을 바꿔넣기할 때 사용된다. 모든 지령들은 지령바꿔넣기를 할 때 거꿀인용부호를 사용한다.[13] bash에는 이를 위한 두가지 형식을 가지고 있는데 하나는 낡은 형태로 지령이 인용부호안에 있을 때이고 새로운 Korn형태는 지령이 $기호가 앞에 있고 그다음 괄호안에 있을 때이다. bash는 지령실행에 의해 확장을 실현하며 행바꾸기코드를 삭제한 지령의 표준출력을 되돌려 보낸다. 바꿔넣기에 낡은 형태를 사용하면 \ 는 $, ' 혹은 \ 을 제외한 정확한 의미만을 보유한다. $형태를 사용하면 모든 문자들은 괄호사이에서 지령을 만들게 된다. 지령바꿔넣기는 틀안에 넣어야 하며 낡은 형식으로 지령을 틀안에 넣기 위해서는 기호와 함께 벗어 나게 해야 한다.

형식

'UNIX 지령'#'기호를 사용한 낡은 방법'
$(UNIX지령)# 새 방법

실례 11-74

(The old Way)

1. $ **echo "The hour is 'date +%H' "**
   The hour is 09

2. $ **name='awk –F: '{print $1}' database'**
   $ **echo $name**
   Ebenezer Scrooge

3. $ **ls 'ls /etc'**

---

13. bash쉘은 앞방향일치성을 위하여 지령바꿔넣기에 거꿀인용부호를 허용하지만 더 좋은 방법을 가지고 있다.

shutdown

4. $ **set 'date'**
5. $ **echo $***
   Wed Jul 14 09:35:21 PDT 2001
6. $ **echo $2 $6**
   Jul 2001
7. $ **echo 'basename \ 'pwd\ "**
   ellie

설명

1. date지령의 출력은 문자렬로 바꿔넣기된다.
2. awk지령의 출력은 변수 name에 대입되여 현시된다.
3. 인용부호안에 있는 ls지령의 출력은 /etc등록부안에 있는 파일목록이다. 파일이름들은 먼저 ls지령을 인수로 한다. 현재등록부 /etc안에 똑 같은 이름으로 된 모든 파일들이 현시된다(이 등록부에 대응되여 있지 않는 파일들은 오유통보문을 내보낸다.).
4. set지령은 위치파라메터로 date지령의 출력을 대입한다. 공백은 파라메터들안에서 단어목록을 구별한다.
6. $*변수는 모든 파라메터들을 가리킨다. date지령의 출력이 $변수안에 기억된다. 매개 파라메터들은 공백으로 구별된다.
6. 두번째와 여섯번째 파라메터들이 현시된다.
7. 현재작업등록부의 이름만을 변수 dirname에 대입하기 위해 지령바꿔넣기가 틀안에 넣어 진다. 먼저 pwd지령이 실행되며 UNIX지령 basename을 인수로 하여 현재작업등록부의 정확한 경로이름이 통과된다. bashname지령은 모든것을 해제하지만 경로이름의 마지막원소는 제외된다. ()안에 지령들을 넣을 때 지령을 위한 인용부호는 \ 과 함께 벗어 나야 한다.

bash가 지령바꿔넣기에서 인용부호를 어떻게 사용하는가를 실례 11-75에서 보여 준다.

실례 11-75

(The New Way)

1. $ **d=$(date)**
   $ **echo $d**
   Wed Jul 14 09:35:21 PDT 2001
2. $ **lines = $(cat filex)**
3. $ **echo The time is $(date +%H)**
   The time is 09
4. $ **machine=$(uname –n)**
   $ **echo $machine**
   jody

5. $ **pwd**
   /usr/local/bin
   $ **dirname="$(basename $(pwd)) "** *# Nesting commands*
   $ **echo $dirname**
   bin
6. $ **echo $(cal)** *# Newlines are lost*
   July 2001 S M Tu W Th F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
   16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
7. $ **echo "$(cal)"**
   July 2001
   S M Tu W Th F S
   1 2 3
   4 5 6 7 8 9 10
   11 12 13 14 15 16 17
   18 19 20 21 22 23 24
   25 26 27 28 29 30 31

설명

1. date지령은 괄호안에 있다. 지령의 출력은 표현식으로 바꿔넣기된 다음 변수 d를 대입하여 현시한다.
2. cat지령의 출력이 변수 lines에 대입된다.
3. 다시 date지령을 괄호안에 넣었다. date+%H의 출력은 표현식에 의해 바꿔넣기된 현재시간이며 현시장치에 현시된다.
4. 변수 machine에 uname -n의 출력으로 주기계의 이름을 대입한다. Machine변수의 값은 현시장치에 출력된다.
5. pwd지령(현재작업등록부)의 출력은 /usr/local/bin이다. 변수 dirname에는 지령바꿔넣기가 틀안에 넣어 지고 실행된 출력을 대입한다. $(pwd)는 우선 실행하려고 하는 지령바꿔넣기이다. pwd지령의 출력이 표현식으로 바꿔넣기된 다음 basename프로그람은 basename/usr/local/bin으로 결과처리된 인수 /usr/local/bin을 바꿔넣기결과로 사용한다.
6. cal지령(현재의 달)출력은 현시된다. 지령바꿔넣기가 이루어 지면 행바꾸기기호가 삭제된다.
7. 2중인용부호안에 모든 지령바꿔넣기표현식을 넣을 때 행바꾸기코드가 보존되며 력서가 보인다.

5. 산수연산확장

쉘은 산수연산표현식을 평가하여 산수연산확장을 실현한 다음 그 결과를 바꿔넣기한다. 표현식은 2중인용부호안에 있으면서 조립되여 있으면 처리된다. 산수연산자의 서술과 연산값계산에 대하여서는 《let지령》에서 보여 준다. 산수연산표현식의 값계산에는 두가지 형태가 있다.

형식

```
$ [표현식]
$ ((표현식))
```

실례 11-76

```
echo $[5 + 4 - 2]
7
echo $[5 + 3 * 2]
11
echo $[(5 + 3) * 2]
16
echo $(( 5 + 4 ))
9
echo $(( 5 / 0 ))
bash: 5/0: division by 0 (error token is "0")
```

6. 확장순서

변수지령, 산수연산표현식 그리고 경로이름을 확장할 때 쉘은 지령행검사를 다음과 같은 순서로 진행한다. 변수들에 인용부호가 없을 때에는 다음의 순서로 진행한다.

1. 대괄호확장
2. ~확장
3. 파라메터확장
4. 변수바꿔넣기
5. 지령바꿔넣기
6. 산수연산확장
7. 단어구분
8. 경로이름확장

7. 배렬(판본 2.x)

bash 2.x판본들은 1차원배렬을 만들수 있다. 배렬은 한개 변수의 이름을 가지고 단어들의 목록 실례를 들면 수들의 목록, 이름들의 목록, 파일들의 목록 등을 대상하기 위해 사용한다. 배렬들은 내부함수 declare –a로 만들어 지거나 변수이름을 보조스크립트로 주는 실례로 x[0]=5와 같이 만들수 있다.

첨수값은 0으로 시작되는 옹근수이다. 묶음에는 최대크기제한이 없으며 수들은 순서대로 배렬되지 않는다. 즉 x[0], x[1], x[2]로만 배렬된다.

배렬가운데서 어떤 원소를 뽑아 내기 위해서는 <${배렬이름[첨수]}>라는 문법을 사용한다. declare –a와 –r추가선택을 주면 읽기전용배렬이 만들어 진다.

형식

declare –a 변수이름
변수 =(항목1, 항목2, 항목3 . . .)

실례 11-77

declare –a nums=(45 33 100 65)
declare –ar names *(array is readonly)*
names=(Tom Dick Harry)
states=(ME [3]=CA CT)
x[0]=55
n[4]=100

배렬에 값을 줄 때 그것들은 자동적으로 첨수 0으로부터 시작하여 매 원소를 첨부할 때마다 첨수가 1만큼 증가한다. 인수의 색인은 제공되지 않으며 만약 하려고 한다면 그것은 순서대로 되지 않는다. 배렬을 해제하기 위해서는 unset지령다음에 묶음이름을 주어야 한다. 또한 묶음가운데서 한개 원소를 해제하기 위해서는 unset하고 묶음이름 [보조스크립트]라는 문법을 사용해야 한다. declare, local 그리고 read_only내부지령들도 배렬을 선언하기 위해서는 –a추가선택을 주어야 한다. read지령에 –a추가선택을 주면 배렬원소들에로 단어목록을 읽어 들일수 있다.

실례 11-78

1. $ **declare - a friends**
2. $ **friends=(Shery1 Peter Louise)**
3. $ **echo ${friends[0]}**
   Shery1
4. $ **echo ${friends[1]}**
   Peter
5. $ **echo ${friends[2]}**
   Louise
6. $ **echo** “All the frinds are **${friends[*]}**”
   All the friends are Shery1 Peter Louise
7. $ **echo** “The number of elements in the array is **${#friends[*]}**”
   The number of elements in the array is 3
8. $ **unset friends** *or* unset ${friends[*]}

설명

1. declare내부지령은 배렬을 명백히 선언하는데 사용되지만 꼭 필요한것은 아니다. 보조스크립트를 사용하는 임의의 변수 실례로 변수[0]에 값을 대입할

때 자동적으로 배렬로 취급된다.
2. 배렬 friends에 값목록 sherly, peter 그리고 louise를 대입한다.
3. friends배렬의 첫번째 원소는 보조스크립트의 값으로 사용되는 첨수값 0과 함께 큰 괄호안에 배렬이름과 보조스크립트를 넣어 줄수 있다. shery가 현시된다.
4. friends배렬의 두번째 원소는 첨수값 1을 써서 대입한다.
5. friends배렬의 세번째 원소는 첨수값 2를 써서 대입한다.
6. 보조스크립트안에 *를 넣으면 배렬의 모든 원소를 호출할수 있다. 이 행은 friends배렬의 모든 원소를 현시한다.
7. 문법 ${#friends[*]}는 묶음의 크기를 지적한다. 즉 배렬안의 원소의 개수이다. 다른 방법으로 ${# friends[0]}는 배렬의 첫번째 원소값에서 문자의 수를 지적한다. Sheryl에서는 6개 문자이다.
8. unset내부지령은 모든 배렬을 지운다. 묶음의 한개 원소는 unset friends[1]을 입력하여 지울수 있다. Sheryl가 지워 진다.

실례 11-79

1. $ **x[3]=100**
   $ **echo ${x[*]}**
   100
2. $ **echo ${x[0]}**
3. $ **echo ${x[3]}**
   100
4. $ **states=(ME [3]=CA [2]=CT)**
   $ **echo ${states[*]}**
   ME CA CT
5. $ **echo ${states[0]}**
   ME
6. $ **echo ${states[1]}**
7. $ **echo ${states[2]}**
   CT
8. $ **echo ${states[3]}**
   CA

설명

1. 묶음의 세번째 원소 x는 150이다. 첨수번호가 3이지만 첫번째, 두번째 요소가 아직 정의되여 있지 않았기때문에 배렬의 크기는 무조건 1이다. 그러므로 ${x[*]}는 묶음 x의 한개 원소를 현시한다.

2. x[1]은 값을 가지지 않으며 또 x[1]과 x[2]도 역시 값을 가지지 않는다.
3. x[3]은 100이라는 값을 가진다.
4. status배렬은 첨수 0에 ME, 첨수 3에 CA 그리고 첨수 2에 CT가 대입된다. 이 실례에서는 bash가 첨수에 어떤 값을 기억시키는가를 상관하지 않으며 함수번호가 순서대로 되지 않는다는것을 보게 된다.
5. states배렬의 첫번째 원소가 현시된다.
6. states[1]에 기억된것이 아무것도 없다.
7. states배렬의 세번째 원소 states[2]에 CT가 대입된다.
8. states배렬의 네번째 원소 states[3]에 CA가 대입된다.

8. 함수소개

bash함수는 현재쉘배경안에서 이름과 함께 지령그룹을 실행시키기 위해 사용된다. 그것들은 스크립트와 같이 매우 효과적이다. 함수는 한번 정의하면 쉘기억기안에 들어가기때문에 호출할 때 파일처럼 자기디스크로부터 읽어 지지 않는다. 함수는 항상 스크립트를 구성하기 위해 사용된다. 함수는 한번 정의하면 계속 사용될수 있다. 함수는 대화형쉘에서 재촉문이 나타나면 정의할수도 있지만 보통 사용자초기화파일 .bash_profile안에서 정의한다. 그것들은 무조건 실행하기전에 정의해야 한다.

함수정의 bash함수를 정의하는데는 두가지가 있다.

한가지 방법은 원래 Bourne쉘에서 쓰는 방법인데 함수이름을 주고 다음에 빈 괄호를 주며 다음에 함수정의를 주는 형식이다.

새로운 방법(Korn쉘에서 쓰는 방법)은 function을 입력한 다음에 함수이름을 주고 그다음 함수정의를 주는 형식이다.

새로운 방법을 사용할 때 괄호는 쓸수도 있고 쓰지 않을수도 있다.

함수정의는 대괄호안에 놓인다. 그것은 반두점으로 구별된 지령들로 구성된다. 제일 마지막지령은 반두점으로 끝나야 한다. 대괄호주위에는 공백이 있어야 한다. 함수에 있는 인수는 함수내에서 위치파라메터로 취급된다. 함수내에 있는 위치파라메터들은 함수내에서만 사용된다. local내부함수는 함수정의에서 만들어 지는것을 국부변수로 대입한다. 함수는 또한 재귀호출할수도 있다. 즉 제한이 없이 계속 자기자신을 호출할수 있다.

형식

```
함수이름(){지령들;지령들;}
function함수이름{함수들;함수들;}
function함수이름(){함수들;함수들;}
```

실례 11-80

1. $ **function greet { echo “Hello $LOGNAME,today is $(date)”;}**
2. $ **greet**

   Hello ellie, today is Wed Jul 14 14:56:31 PDT 2001

3. $ **greet () { echo "Hello $LOGNAME,today is $(date)";}**
4. $ **greet**

   Hello ellie,today is Wed Jul 14 15:16:22 PDT 2001
5. $ **declare –f**

   declare –f greet()

   {

   echo "Hello $LOGNAME, today is $(date)"

   }
6. $ **declare –F**[ㄱ]

   declare –f greet
7. $ **export –f greet**
8. $ **bash** *Start sub shell*
9. $ **greet**

   Hello ellie,today is Wed Jul 14 17:59:24 PDT 2001

설명

1. 예약어 function다음에 함수이름 greet가 있다. 함수정의는 { }기호안에 있다. 대괄호를 연 다음에는 공백을 주어야 한다. 같은 행우에서 명령문들은 반두점으로 끝나야 한다.
2. greet함수를 실행할 때 중괄호안에 있는 지령들은 현재쉘배경에서 실행된다.
3. greet함수는 Bourne쉘문법 즉 함수이름다음에 빈 괄호를 주고 함수정의를 주는 형식을 사용하여 다시 정의된다.
4. greet함수가 다시 실행된다.
5. declare지령에 –f추가선택을 주면 쉘에서 정의된 모든 함수들과 정의내용들을 현시한다.
6. declare지령에 –f추가선택을 쓰면 함수이름들만 현시된다.
7. export지령에 –f추가선택을 쓰면 일반함수를 작성한다. 즉 쉘을 쓸수 있다.
8. 새로운 bash쉘이 기동된다.
9. 함수는 반출되였기때문에 자식쉘로 정의된다.

ㄱ. bash 판본 2.x에서만 사용한다.

실례 11-81

1. $ **function fun {**

```
        echo "The current working directory is $PWD."
        echo "Here is a list of your files: "

        ls
        echo "Today is $(date +%A).";
    }
```

2. $ **fun**

   The current working directory is /home.

   Here is a list of your files:

   abc abc123 filel.bak none nothing tmp

   abc1 abc2 file2 nonsense nowhere touch

   abc122 file1 file2.bak noone one

   Today is Wednesday.

3. $ **function welcome { echo "Hi $1 and $2"; }**
4. $ **welcome tom joe**

   Hi tom and joe

5. $ **set jane anna lizzy**
6. $ **echo $***

   jane anna lizzy

7. $ **welcome johan joe**

   hi johan and joe

8. $ **echo $1 $2**

   johan joe

9. $ **unset –f welcome** *# unsets the function*

설명

1. 함수 fun이 이름 지어 지고 정의된다. 예약어 function다음에 함수이름과 대괄호안에 지령들의 목록이 차례로 들어 있다. 지령들은 서로 다른 행우에 현시하며 같은 행에 현시하려면 반두점으로 분리해야 한다. 공백은 첫번째 중괄호다음이나 문법오유를 얻으려고 할 때 필요하다. 함수는 사용되기전에 정의되여야 한다.
2. 함수는 실행될 때 스크립트처럼 처리된다. 함수정의안에 있는 모든 지령들은 그 함수를 실행할 때 실행된다.
3. 함수 welcome안에는 2개의 위치파라메터가 있다. 함수에 인수를 줄 때 위치파라메터에는 값이 대입된다.
4. 함수 tom과 joe의 인수는 실지로 $1와 $2에 대입된다. 함수안에 있는 위치

파라메터들은 함수전용으로서 그외에는 관계가 없다.
5. 위치파라메터들은 지령행에서 대입된다. 이 변수들은 함수안에서 하나씩 대입하여도 된다.
6. $*는 현재 대입된 위치파라메터들의 값들을 현시한다.
7. 함수 welcome의 johan와 joe는 위치파라메터를 대입한 값이다.
8. 지령행에서 대입한 위치변수 $들을 함수안에서 대입하면 효력이 없어 진다.
9. unset내부지령에 –f추가선택을 사용하면 함수를 해제한다. 짧게 정의된다.

함수의 현시와 해제 함수들과 정의들을 현시하기 위해서는 declare지령을 사용한다. bash판본 2.x이상에서 declare –F지령은 함수이름들을 현시한다. 함수와 정의는 국부변수들을 반출하는데 따라 출력에 나타난다. 함수와 정의는 unset –f지령에 의해 해제된다.

9. 표준 I/O와 방향바꾸기

쉘이 기동하면 3개의 파일 즉 표준입력, 표준출력 그리고 표준오유를 제공한다. 표준입력은 보통 건반으로, 표준출력과 표준오유는 현시장치로 한다. 그것들은 모두 파일로부터 입력하거나 파일에로 출력하거나 오유통보를 보낼 때 사용한다. 그것들은 I/O방향바꾸기를 사용하여 변경할수 있다.

방향바꾸기연산자목록을 표 11-23에 보여 준다.

표 11-23. 방향바꾸기

방향바꾸기연산자	의 미
<	입력방향바꾸기
>	출력방향바꾸기
>>	출력첨가
2>	오유방향바꾸기
&>	출력과 오유방향바꾸기
>&	출력과 오유방향바꾸기(더 좋은 방법이다.)
2>&1	출력하면서 오유방향바꾸기
1>&2	오유로 내보내면서 출력방향바꾸기
>1	출력방향바꾸기할 때 noclobber를 무시한다.
<>파일이름	장치파일(/dev로부터)이면 표준입력와 표준출력에 모두 파일을 사용한다.

실례 11-82

```
1.  $ tr '[A-Z]' '[a-z]' < myfile   # Readirect input
2.  $ ls > lsfile                   # Redirect output
    $ cat lsfile
```

dir1

dir2

file1

file2

file3

3. $ **date >> lsfile** *# Redirect and append output*

    $ cat lsfile

    dir1

    dir2

    file1

    file2

    file3

    Sun Sept 17 12:57:22 PDT 2001

4. $ **cc prog.c 2 >errfile** *# Readirect error*

5. $ **find . –name \ *.c –print > foundit 2> /dev/null**

    # Redirect output to foundit and errors to /dev/null,

    # respectively.

6. $ **find . –name \ *.c –print > & foundit**

    #Redirect both output and errors to oundit.

7. $ **find . –name \ *.c –print > foundit** 2 ⌒**&** 1

    # Redirect output to foundit and send errors to where output

    # is going; i.e. foundit

8. $ **echo “File needs an argument” 1>&2**

    #Send standard output to error

설명

1. 표준입력은 UNIX tr지령을 건반으로부터 입력할 대신에 파일 myfile로부터 실행되도록 방향을 바꾼다. 모든 큰 문자들을 작은 문자로 바꾼다.
2. 현시장치에 출력을 보낼 대신에 ls지령이 파일 lsfile에 출력되도록 방향을 바꾼다.
3. date지령의 출력이 방향바꾸어 지고 파일 lsfile에로 첨가된다.
4. C프로그람원천파일 prog.c가 번역된다. 번역이 실패하면 표준오유가 errfile으로 방향바꾸기된다.
5. find지령은 *.c로 편성한 파일들을 현재작업등록부에서 검사하기 시작한다.

그다음 foundit라는 파일에 파일이름들을 써넣는다. find지령의 오유는 /dev/null에 보내진다.
6. find지령은 *.c로 편성한 파일이름들을 현재작업등록부에서 검사하기 시작한다. 그다음 foundit라는 파일에 파일이름들을 써넣는다. 오유도 역시 foundit파일에 보낸다.
7. 6과 같다.
8. echo지령은 표준오유에 통보문을 낸다. 표준출력은 표준오유와 같이 설정되여야 한다.

exec지령과 방향바꾸기 exec지령은 새로운 프로쎄스를 기동함이 없이 새롭게 현재 프로그람을 바꿔넣기하는데 사용할수 있다. 표준출력과 입력은 자식쉘을 구성함이 없이 exec지령에 의해 변경시킬수 있다(표 11-24에서 보여 준다.).

만약 파일을 exec로 열기할 때 read지령은 파일의 끝이 아니라면 파일지시기를 움직일수 있다. 파일을 다시 시작하여 읽으려면 닫아야 한다. cat와 sort와 같은 UNIX지령들을 사용하면 조작체계는 매 지령을 처리한 다음 파일을 닫는다.

표 11-24. exec지령

exec지령	의 미
exec ls	ls지령은 쉘이 있는데서 실행된다. ls가 끝날 때 그것을 기동한 쉘은 귀환하지 않는다.
exec<filea	표준입력을 읽기 위해 filea를 연다.
exec>filex	표준출력을 쓰기 위해 filex를 연다.
exec 3<datfile	입력을 읽기 위해 파일서술자 3으로 datfile을 열기한다.
sort<&z	datfile이 분류된다.
exec 4>newfile	쓰기 위해 파일서술자(fd) 4로 newfile을 열기한다.
ls>&4	ls의 출력은 newfile으로 방향바꾸기된다.
exec5<&4	fd4의 복사를 fd5로 한다.
exec 3<&-	fd3을 닫는다.

실례 11-83

1. $ **exec date**
   Thu Oct 14 10:07:34 PDT 2001
   <Login prompt appears if you are in your login 쉘 >
2. $ **exec > temp**
   $ **ls**
   $ **pwd**
   $ **echo Hello**

3. $ **exec > /dev/tty**
4. $ **echo Hello**
   Hello

설명

1. exec지령은 현재쉘안에서 date지령을 선택한다. Date지령은 현재쉘에서 실행되기때문에 date지령을 탈퇴할 때 쉘이 종결된다. bash쉘이 TC쉘로부터 기동되면 bash쉘은 탈퇴하고 TC쉘재촉문이 나타난다. 이때 등록가입쉘상태이면 등록해제된다. 쉘창문안에서 작업하고 있으면 창문이 닫긴다.
2. exec지령은 현재쉘표준출력으로 temp파일을 연다. ls, pwd 그리고 echo지령들의 출력은 temp파일로 된다(그림 11-3).
3. exec지령은 말단을 표준출력으로 다시 연다. 현재출력은 4행에서 보는바와 같이 화면으로 된다.
4. 표준출력은 말단(/dev/tty)을 다시 방향바꾸기한다.

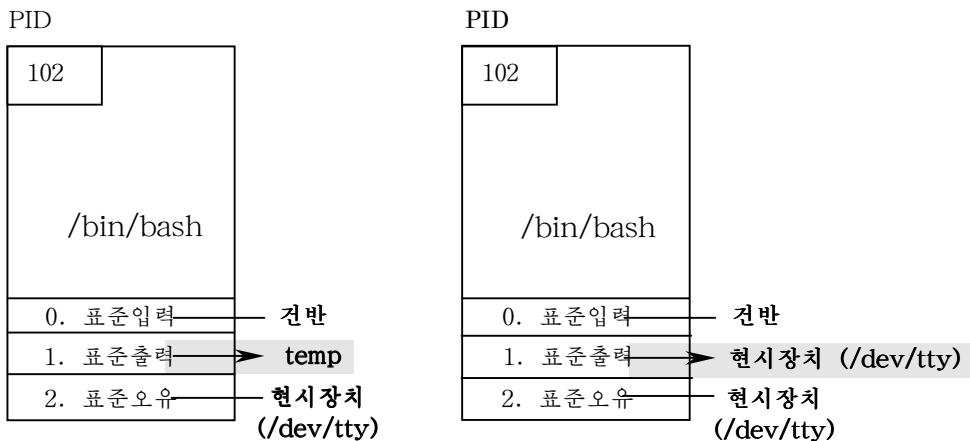



그림 11-3. exec 지령과 파일서술자

실례 11-84

1. > **bash**
2. $ **cat doit**
   pwd
   echo hello
   date
3. $ **exec < doit**
   /home/homebound/ellie/쉘
   hello
   Thu Oct 14 10:07:34 PDT 2001
4. >

설명

1. TC쉘재촉문으로부터 bash가 기동된다(이것은 exec지령이 탈퇴될 때 일어나며 사용자는 가입해제되지 않는다.).
2. doit파일의 내용이 현시된다.
3. exec지령은 doit파일을 표준입력으로 열기한다. 입력은 건반대신에 파일로부터 읽어 진다. Doit파일로부터 넘겨 받는 지령들은 현재쉘에서 실행된다. 마지막지령프로쎄스를 끝내면 곧 쉘이 동작한다.
4. bash쉘은 exec지령을 끝마칠 때 끝난다. TC쉘재촉문이 나타난다. 어미쉘로 된다. exec를 끝마칠 때 등록가입쉘상태이면 등록해제된다. 창문안에 있으면 창문에 현시된다.

실례 11-85

1. $ **exec 3 > filex**
2. $ **who >& 3**
3. $ **date >& 3**
4. $ **exec 3>&-**
5. $ **exec 3<filex**
6. $ **cat <&3**

   ellie tty1 Jul 21 09:50

   ellie ttyp1 Jul 21 11:16 (:0.0)

   ellie ttyp0 Jul 21 16:49 (:0.0)

   Wed Jul 21 17:15:18 PDT 2001
7. $ **exec 3<&-**
8. $ **date >& 3**

   date:write error: Bad file descriptor

설명

1. 파일서술자 3(fd 3)에 filex를 대입한다. 그리고 출력방향바꾸기를 위해 연다(그림 11-4(ㄱ)에서 보여 준다.).
2. who지령의 출력을 fd3 즉 filex에 보낸다.
3. date지령의 출력을 fd3에 보낸다. Filex는 이미 열어 졌다. 따라서 출력은 filex에 첨가된다.
4. fd3이 닫긴다.
5. exec지령을 입력하기 위해 fd3을 연다. 입력은 filex로 방향바꾸기되였다. 그림 11-4(ㄴ)에서 보여 준다.
6. cat지령은 fd3으로부터 읽는다. Filex를 허용한다.
7. exec지령은 fd3을 닫는다 (실제로 조작체계는 다만 파일의 끝에 도달한 파일만을 닫는다.).
8. fd3에 date지령의 출력을 보내려고 할 때 bash는 fd3이 이미 닫겨 져 있기때문에 오유조건을 알려 준다.

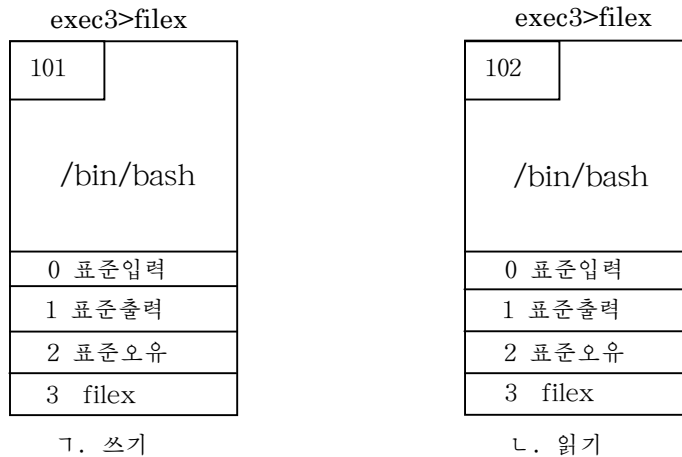



그림 11-4. exec 와 파일서술자

10. 파이프

파이프는 파이프의 왼쪽에 있는 지령으로부터 출력을 받아 가지고 파이프의 오른쪽에 있는 지령에 입력을 보낸다. 파이프행은 여러개의 파이프로 이루어 질수 있다.

실례 11-86의 지령들은 등록가입하는 사용자수를 계수하며 파일(tmp)안에 지령의 출력을 기억하고 다음 tmp파일안에 행수를 계수하여 넣기 위해 wc-1을 사용한 다음 tmp파일을 지운다(즉 등록가입된 사용자수를 찾는다.).

실례 11-86

```
1. $ who   > tmp
2. $ wc –1 tmp
   4 tmp
3. $ rm tmp

   # Using a pipe saves disk space and time.

4. $ who | wc –1
   4

5. $ du .. | sort –n |sed –n ‘$p’
   1980      ..

6. $ ( du / | sort –n | sed –n ‘$p’ ) 2> /dev/null
   1057747 /
```

설명

1. who지령의 출력이 tmp파일로 방향바꾸기된다.
2. wc-1지령은 tmp파일안에 있는 행수를 현시한다.
3. tmp파일을 지운다.

4. 파이프하면 매 단계는 3단계로 실행할수 있다. who지령의 출력은 핵심부 완충기로 보내며 wc-l 지령을 완충기로부터 읽고 현시장치에 출력을 보낸다. 그림 11-5에서 보여 준다.
5. 어미등록부로부터 시작하여 매 등록부당 사용되는 디스크블로크들의 수인 du지령의 출력은 sort지령에 의해 파이프되여 수값적으로 분류된다. 그다음 자기가 받은 출력의 마지막행을 현시하는 sed지령과 파이프된다. 그림 11-6에서 보여 준다.
6. du지령(뿌리등록부에서 시작한다.)이 허가되지 않아 등록부안에 기입될수 없으면 stderr(현시장치)에 오유통보문을 내보낸다. 괄호안에 전체 지령행을 넣으면 모든 출력이 현시장치에 현시되고 모든 오유는 UNIX비트묶음 /dev/null로 방향바꾸기한다.

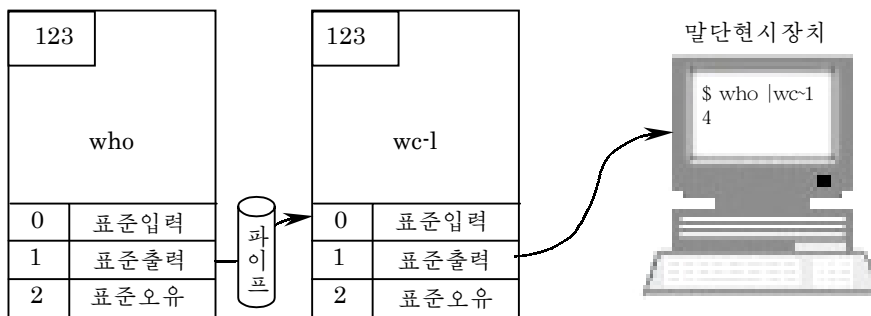



그림 11-5. 파이프

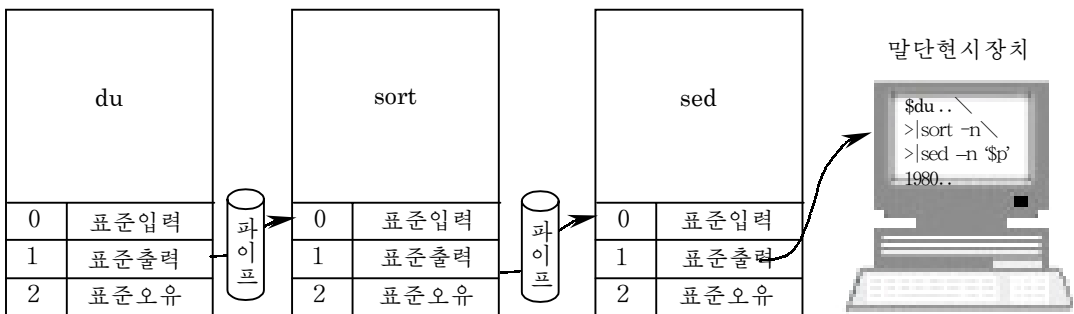



그림 11-6. 다중파이프(려파기)

11. here문서와 입력방향바꾸기

here문서는 인용을 위한 규정형태이다. 그것은 사용자정의최종완료자가 도달할 때까지 mail, sort 혹은 cat와 같은 입력대기프로그람을 위한 열린 본문을 접수한다. 이것들은 차림표들을 작성하기 위해 쉘스크립트에서 자주 사용된다.

입력을 넘겨 받는 지령은 《기호다음에 사용자정의단어나 혹은 기호가 오며 다음에 행바꾸기가 첨부된다. 입력은 사용자정의단어나 혹은 기호가 왼쪽 제일 마지막렬안에 놓

이면 완결된다. 사용자정의단어는 입력을 중지하기 위해 Ctrl-D를 눌러야 하는 위치에서 사용된다. 만약 최종완료자가 <<-연산자, 끌기타브 그리고 타브만을 앞에 놓으면 마지막 최종완료자를 앞에 놓아야 한다. 사용자정의최종완료자나 기호는 실지로는 <<here>>로부터 <<here>>까지 대조한다.

다음의 실례들은 문법을 설명하기 위해 지령행에 here문서를 사용한것이다. 스크립트안에서 사용하면 아주 효과적이다.

실례 11-87

```
1. $  cat << FINISH                    # FINISH is a user-defined
2. >  Hello there $LOGNAME               # terminator
3. >  The time is $(date   +%T).
   >  I can't wait to see you !!!
4. >  FINISH                            # terminator matches first

5.  Hello there ellie                     # FINISH on line 1.
   The time is 19:42:12.
   I can't wait to see you !!!
6.  $
```

설명

1. UNIX cat프로그람은 단어 FINISH가 행우에 나타날 때까지 입력을 허용한다.
2. 2차재촉문이 나타난다. 그다음 본문은 cat지령을 위한 입력이다. 변수바꿔넣기는 <here문서>안에서 실행된다.
3. 지령바꿔넣기 즉 $(date+%T)는 here문서안에서 실행된다. 또한 지령바꿔넣기문서형태를 사용한다. 즉 'date+T'이다.
4. 사용자정의최종완료자 FINISH는 cat프로그람의 입력끝을 현시한다. 그 앞뒤에 아무런 공백도 가질수 없고 자기 행우에 있어야 한다.
5. cat프로그람으로부터 출력이 현시된다.
6. 쉘재촉문이 다시 나타난다.

실례 11-88

```
1. $  cat <<- DONE
   >    Hello there
   >          What's up?
   >Bye now The time is 'date'.
2. >      DONE
   Hello there
   What's up ?
   Bye now The time is Sun Feb 819:48:23 PST 2001.
   $0
```

설명

1. cat프로그람은 DONE이 자체행우에 나타날 때 입력을 허용한다. <<-연산자는 한개이상의 타브들이 선행되도록 입력과 최종완료자를 허용한다. 지령행에서 이 실례를 입력하는것은 타브건에 문제가 있다. 만약 스크립트로부터 실행하면 실례는 정확한 조작이다.
2. 마지막대응최종완료자 DONE는 타브를 실행시킨다. cat프로그람의 출력이 현시장치에 현시된다.

12. 쉘호출추가선택

쉘이 bash지령을 사용하여 기동될 때 그의 특성을 변경시키기 위한 추가선택이 있다. 추가선택에는 2가지 형태 즉 단일문자추가선택들과 다중문자추가선택들이 있다. 단일문자추가선택은 한개의 –(횡선)과 단일문자로 구성된다. 다중문자추가선택은 2개의 --(련결기호)와 임의의 물음표들로 구성된다.

다중문자추가선택들은 단일문자추가선택전에 대입해야 한다. 대화형등록가입쉘은 보통 –I(대화형쉘을 기동한다.), -s(표준입력으로부터 읽는다.) 그리고 –m(일감조종이 가능하다.)등과 함께 기동한다. 표 11-25에서 보여 준다.

표 11-25. bash 2.x쉘호출추가선택

추가선택	의 미
-c문자렬	지령들은 문자렬로부터 읽어 진다. 문자렬다음의 임의의 인수들에는 $0으로 시작되는 위치파라메터들을 대입한다.
-D	$다음에 오는 2중인용부호물음표목록은 표준출력에 출력된다. 이 문자렬은 현재상태가 c나 혹은 POSIX가 아닐 때 언어에 해당된다. –n추가선택은 함축되며 지령이 아닌것이 실행된다.
-i	쉘이 대화형상태에 있다. TERM, QUIT 그리고 INTERRUPT는 무시된다.
-s	지령은 표준출력로부터 읽어 지며 위치파라메터문자렬을 허용한다.
-r	제한된 쉘이 기동한다.
- -	추가선택들의 끝을 표기하며 제일 마지막추가선택을 무효로 한다. –-와 –다음의 모든 인수들은 파일이름과 인수들로 처리된다.
- - dump-strings	-D와 같다.
- - help	내부지령과 탈퇴들을 위한 사용통보문을 현시한다.
- - login	등록가입쉘로 bash가 되게 한다.
- - noediting	bash가 대화형으로 집행될 때 readline서고를 사용하지 않는다.
- - noprofile	bash가 대화형으로 실행될 때 읽기행서고를 사용하지 않는다. 기동할 때 bash 즉 /etc/profile, ~/.bash_profile, ~/.bash_login 혹은 ~/.profile을 읽지 않는다.

(표계속)

추가선택	의 미
- - norc	대화형쉘들을 위해 bash는 ~/.bashrc파일을 읽지 않는다. 기동은 쉘이 실행될 때 대입되여 있다.
- - posix	필요에 따라 POSIX 1003.2표준에 맞게 bash의 특성을 변경한다.
- - quiet	쉘기동에 비정보들을 현시한다. 기동으로 대입된다.
- - rcfile file파일	bash가 대화형이면 ~/.bashrc초기화파일대신에 이 초기화파일을 사용한다.
- - restricted	제한된 쉘을 기동한다.
- - verbose	-v와 같다. 확장방식을 설정한다.
- - version	bash쉘과 탈퇴에 대한 판본정보를 현시한다.

13. set지령과 추가선택

set지령은 지령행인수들을 잘 처리하도록 하는 쉘추가선택들을 설정/해제하기 위해 사용된다. 추가선택을 대입하려면 –기호를 추가선택앞에 써주며 추가선택을 해제하려면 +기호를 써주어야 한다. set추가선택목록을 표 11-26에 보여 준다 .

실례 11-89

1. $ **set –f**
2. $ **echo ***
3. $ **echo ??**
   ??
4. $ **set +f**

설명

1. f추가선택이 설정되므로 파일이름확장이 무효로 된다.
2. *는 확장되지 않는다.
3. 물음표(?)표식은 확장되지 않는다.
4. f추가선택이 해제되며 파일이름확장이 가능하다.

표 11-26. set**내부지령추가선택**

추가선택이름	지름추가선택	의 미
allexport	-a	자동적으로 추가선택이 대입된 때로부터 해제될 때까지 반출되는 새로운 변수들이나 혹은 변경된 변수들을 현시한다.
braceexpand	-B	괄호확장을 할수 있다. 기정으로는 설정이다.
emacs		지령행편성에 emacs내부편집기를 사용한다. 기정으로는 설정이다.

(표계속)

추가선택이름	지름추가선택	의 미
errexit	-e	지령에 0아닌 탈퇴상태값(실패)을 되돌려 주면 탈퇴한다. 초기화파일을 읽을 때 대입되지 않는다.
histexpand	-H	리력바꿔넣기를 실행할 때 !와 !!를 쓸수 있다.기정으로는 설정이다.
history		지령행리력을 쓸수 있다. 기정으로는 설정이다.
ignoreeof		쉘은 탈퇴할 때 EOF(ctrl-D)를 무효로 한다. 그러므로 exit라고 입력해야 한다. 쉘변수 IGNOREEOF=10으로 대입한것과 같다.
keyword	-k	지령을 위한 환경안에 예약어인수들을 넣는다.
interactive-comments		대화형쉘에 대해 #기호는 행우에 써넣어 진다. 임의의 본문을 설명문으로 사용한다.
monitor	-m	일감조종을 허락한다.
noclobber	-C	방향바꾸기를 사용할 때 파일겹쳐쓰기로부터 파일을 보호한다.
noexec	-n	지령들을 읽지만 실행은 하지 않는다. 스크립트문법검사에 사용한다. 대화형쉘이 작용할 때에는 설정되지 않는다.
noglob	-d	경로이름확장을 무효로 한다.
notify	-b	배경일감이 끝날 때 사용자를 통지한다.
nounset	-u	설정되여 있지 않는 변수를 확장할 때 오유를 현시한다.
onecmd	-t	한개 지령을 읽고 실행한 다음 탈퇴한다.
physical	-P	설정되여 있으면 cd 혹은 pwd를 입력할 때 기호련결은 하지 않는다. 그대신 물리적인 등록부가 사용된다.
posix		쉘특성의 기정조작이 posix표준으로 되여 있지 않으면 변경된다.
privileged	-p	설정되였을 때 쉘은 profile이나 혹은 ENV파일을 읽지 않으며 쉘들은 환경으로부터 이어 지지 않는다.
verbose	-v	오유수정을 위해 확장방식을 설정한다.
vi		지령행편집을 위해 vi내부편집기를 사용한다.
xtrace	-x	오유수정을 위해 현시방식을 설정한다.

14. shopt지령과 추가선택

shopt(bash 2.x)지령은 또한 쉘추가선택들을 설정 혹은 해제하기 위해 사용한다.

표 11-27. shopt지령추가선택

추가선택	의 미
cdable-vars	cd내부지령의 인수가 등록부가 아니면 변수의 값이 등록부가 되는 변수의 이름을 대입한다.
cdspell	cd지령의 등록부이름의 맞춤법오유를 수정한다. 오유는 서로 바뀌여 진 문자들,빠진 문자들 그리고 문자가 너무 많을 때 검사된다. 정확히 오유를 검출하면 경로를 출력하며 지령을 처리한다. 이것은 다만 대화형쉘에서만 쓰인다.
checkhash	bash는 실행하기전에 하쉬표에서 지령이 있는가를 검사한다. 최근에 사용된 지령들이 보통 검색된다.
checkwinsize	bash는 echo지령을 실행한 다음에 창문크기를 검사한다. 필요하면 행과 렬의 값을 갱신한다.
cmdhist	bash는 똑같은 리력입구점에서 모든 지령행들을 기억하기 위해 현시된다. 이것은 다중행지령들을 쉽게 재편성하기 위해 사용된다.
dotglob	Bash는 파일이름확장결과안에 점(.)으로 시작하는 파일이름을 포함한다.
execfail	비대화형쉘은 exec내부지령의 인수로 규정된 파일을 실행시킬수 없으면 탈퇴하지 않는다. 대화형쉘은 exec가 실패하면 탈퇴하지 않는다.
expand-aliases	변경이 확장된다. 확장된 패턴은 유효하다.
extglob	확장된 패턴은 특성이 대응되게 할수 있다(규칙확장메타문자를 위해 Korn쉘로부터 유도된다.).
histappend	쉘은 탈퇴할 때 파일을 겹쳐쓰기하는것이 아니라 HISTFILE에 의해 대입된 파일에로 첨가된다.
histreedit	readline을 사용하고 있으면 실패한 리력바꿔넣기를 재편성하기 위한 기회를 사용자에게 준다.
histverify	Readline이 설정되여 사용중에 있으면 리력바꿔넣기의 결과는 쉘해석프로그람을 통과하지 않는다. 그대신 결과행은 readline편성완충기안에 적재되면서 그이상의 변경을 허락한다.
hostcomplete	Readline이 설정되여 사용중에 있으면 bash는 @를 포함하는 단어가 완성될 때 주이름완성을 실행하도록 한다. 기정으로는 유효이다.
huponexit	Bash가 설정되여 대화형등록가입쉘을 탈퇴할 때 모든 일감은 SIGHUP를 보낸다.
Interactive-comments	대화형쉘에서 무시되는 행우에 있는 문자들을 대신하여 #로 시작하는 단어를 허용한다. 기정으로는 유효이다.
lithist	이 추가선택과 cmdhist추가선택을 허용하면 다중행지령들은 반두점구분기호를 사용하지 않고 행바꾸기와 함께 리력에 기억된다.
mailwarn	설정되면 bash가 전자우편을 검사하고 있는 파일은 그것이 마지막으로 검사한 순간부터 호출된다. 통보문 《mailfile안에 있는 전자우편이 읽어 졌다.》가 현시된다.

(표계속)

추가선택	의 미
nocaseglob	설정하면 bash는 파일확장이 끝난 다음 조건처리에 의해 파일이름을 대조한다.
nullglob	설정하면 bash는 문자렬을 확장하기 위해 파일이 아니라 파일이름패턴들을 처리한다. 기정으로는 유효이다.
promptvars	설정하면 재촉문문자렬은 확장된 다음 변수와 파라메터확장을 진행한다.
restricted_shell	제한방식으로 기동하면 쉘은 이 추가선택을 설정한다. 값은 변경되지 않는다. 이것은 기동파일들을 실행할 때 다시 설정되지 않는다. 쉘이외의것들이 제한되는것을 발견하기 위해 기동파일들을 허용한다.
shift_verbose	이것을 설정하면 shift내부지령은 밀기계수가 위치파라메터개수를 초과할 때 오유통보문을 인쇄한다.
sourcepath	설정하면 source내부지령은 인수로 제공되는 파일을 포함하는 등록부를 찾기 위해 PATH의 값을 사용한다. 기정으로는 유효이다.
source	점(.)과 같다.

15. 쉘내부지령

쉘은 원천코드를 내장하고 있는 지령들을 가지고 있다. 지령들이 내장되여 있기때문에 쉘은 디스크에 놓이지 않고 고속으로 실행된다. help지령은 임의의 내부지령에 대하여 실시간방조를 준다. 내부지령들은 표 11-28에 보여 준다.

표 11-28. 내부지령

지 령	의 미
:	아무런 동작도 하지 않는다. 탈퇴상태 0을 준다.
. file	점(.)지령을 읽고 파일로부터 지령을 실행한다.
break[n]	703페지의 《순환조종》에서 보여 준다.
.	현재프로쎄스의 배경안에서 프로그람을 실행한다. Source와 같다.
alias	존재하는 지령들을 《별명》으로 표시하고 작성한다.
bg	배경에 일감을 넣는다.
bind	현재건과 함수묶음들 혹은 readline함수나 마크로를 건들로 묶은것을 현시한다.
break	제일 마지막순환의 출력을 중지한다.
builtin[sh- builtin [args]]	인수들을 넘기고 탈퇴상태 0을 되돌려 주면서 쉘내부지령을 실행한다. 보통 함수와 내부지령에 꼭 같은 이름이 붙여 졌으면 그렇게 된다.
cd[arg]	Arg가 없으면 home등록부에로 그렇지 않으면 arg의 값으로 변경한다.

(표계속)

지 령	의 미
Command command [arg]	함수가 같은 이름으로 되여 있으면 즉 함수순환대입에 따라 지령을 집행한다.
continue[n]	703페지의 《순환조종》에서 보여 준다.
declare[var]	모든 변수 또는 모든 선언된 변수들은 추가선택속성과 함께 현시된다.
dirs	pushd로 얻어 진 현재등록부목록을 현시한다.
disown	일감표로부터 활동일감을 소거한다.
echo[args]	행바꾸기코드로 끝나는 변수들을 현시한다.
enable	쉘내부지령을 유효 또는 무효로 한다.
eval[args]	쉘을 입력으로 하여 변수들을 읽고 얻어 진 지령들을 실행한다.
exec command	현재쉘위치에서 지령을 실행한다.
exit[n]	탈퇴상태 n으로 쉘을 탈퇴한다.
export[var]	자식쉘로 알려 진 변수를 표식한다.
fc	편집된 리력지령을 위한 리력고정지령
fg	전경에 배경일감을 넣는다.
getopts	지령행추가선택들을 해석하고 처리한다.
hash	지령들을 고속검색하기 위해 내부하쉬표를 조종한다.
help[command]	내부지령에 대한 방조통보를 현시하며 지령이 규정되여 있으면 내부지령에 대하여 구체적으로 설명한다.
history	행번호와 함께 리력목록을 현시한다.
jobs	배경안에 넣은 일감들을 현시한다.
kill[-signal process]	프로쎄스의 PID번호나 혹은 일감번호에 신호를 보낸다. 재촉문이 나타났을 때 kill-1이라고 입력한다.
getopts	지령들을 해석하기 위해 쉘스크립트에서 사용하며 정확한 추가선택들인가를 검사한다.
let	산수연산표현식의 값을 계산하는데 사용하며 변수에 산수연산계산값을 대입한다.
local	함수에서 변수의 적용범위를 결정하는데 사용한다.
logout	등록가입쉘에서 탈퇴한다.
popd	등록부탄창으로부터 입구점들을 소거한다.
pushd	등록부탄창에로부터 입구점들을 첨부한다.
pwd	현재작업등록부를 출력한다.
read[var]	인수 var에 표준입력장치로부터 행을 읽어 넣는다.
readonly[var]	변수 var를 읽기전용으로 표식한다. 다시 설정할수 없다.
return[n]	탈퇴값 n으로 함수를 탈퇴한다.

(표계속)

지 령	의 미
set	추가선택들과 위치파라메터를 대입한다(667페지의 《set지령과 위치파라메터들》에서 보여 준다.).
shift[n]	왼쪽으로 n번 위치파라메터를 옮긴다.
stop pid	프로쎄스번호 PID의 실행을 중지한다.
suspend	현재쉘의 실행을 중지한다(하지만 등록가입쉘이면 안된다.).
test	파일형태들을 검사하고 조건식을 평가한다.
times	이 쉘로부터 실행되는 프로쎄스에 대한 사용자와 체계시간들을 출력한다.
trap[arg][n]	쉘이 신호 n[0, 1, 2 혹은 15]을 받을 때 arg를 실행한다.
type[command]	지령의 형태이다. 실례로 <pwd는 쉘내부지령이다.>를 출력한다.
typeset	declare와 같다. 변수들을 대입하고 속성들을 준다.+
ulimit	프로쎄스자원한계를 현시하고 대입한다.
umask[octal digits]	자기자체와 그룹 기타 그밖의것들과 관련한 파일방식마스크를 설정한다.
unalias	별명들을 해제한다.
unset[name]	변수의 값 혹은 함수를 해제한다.
wait[pid # n]	PID번호 n을 가진 배경프로쎄스를 대기하며 완료상태를 알려 준다.

Bash쉘련습

련습문제 43: 기동

1. 어떤 프로쎄스가 현시장치에 등록가입재촉문을 현시하는가?
2. 어떤 프로쎄스가 HOME, LOGNAME 그리고 PATH에 값을 대입하는가?
3. 어떤 쉘이 사용되고 있는가? 어떻게 알수 있는가?
4. 어떤 지령이 등록가입쉘을 변경하는것을 허락하는가?
5. 지정된 등록가입쉘이 어디에 있는가?
6. /etc/profile과 ~/.bash_profile파일의 프로쎄스를 설명하시오. 어느것이 먼저 실행되는가?
7. 다음과 같은 bash_profile을 편성하시오.

   ㄱ. 사용자를 환영한다.

   ㄴ. home등록부가 경로에 없으면 거기에 추가하시오.

   ㄷ. stty를 사용하여 공백건을 지우기로 설정하시오.

   ㄹ. source.bash-profile을 건입력하시오.
   source지령의 기능은 무엇인가?

8. BASH_ENV파일은 무엇인가? 언제 실행되는가?
9. 기정으로 설정된 본래의 재촉문은 무엇인가?
   ㄱ. 시간과 home등록부를 포함하도록 재촉문을 변경하시오.
   ㄴ. 기정으로 설정된 두번째 재촉문은 무엇인가? 그의 기능은 무엇인가?
10. 다음의 모든 기능을 설명하시오.
   ㄱ. set –o ignoreeof
   ㄴ. set –o noclobber
   ㄷ. set –o emacs
   ㄹ. set –o vi
11. 앞의 실례에서 설정들은 어떤 파일에 기억되는가?
    왜 그것들은 거기에 기억되는가?
12. shopt -p는 무엇을 하는가? 왜 set대신 shopt를 사용하는가?
13. 내부지령이란 무엇인가? 지령이 내부지령인가 혹은 실행가능한 지령인가를 어떻게 설명할수 있는가? builtin지령의 목적은 무엇인가? enable지령의 목적은 무엇인가?
14. 어떻게 쉘이 탈퇴상태값 127을 되돌리는가?

련습문제 44: 일감조종

1. 프로그람과 프로쎄스사이에는 어떤 차이가 있는가? 일감이란 무엇인가?
2. 사용자쉘의 PID는 무엇인가?
3. 일감을 어떻게 중지하는가?
4. 무슨 지령이 배경일감들을 전경일감으로 넘겨 주는가?
5. 실행되고 있는 모든 일감들을 어떻게 표시하는가? 정지된 모든 일감들은?
6. kill지령의 의미는 무엇인가?
7. jobs-1은 무엇을 표시하는가? kill-1은 무엇을 표시하는가?

련습문제 45: 지령완성과 리력 그리고 별명

1. 파일이름완성이란 무엇인가?
2. 지령행에 입력된 지령들의 리력을 기억하는 파일이름은 무엇인가?
3. HISTSIZE변수는 무엇을 조종하는가?
   HISTFILESIZE변수는 무엇을 조종하는가?
4. 리력문자의 의미는 무엇인가?
5. v로 시작되는 제일 마지막지령을 어떻게 재실행하는가?
6. 125번째 지령을 어떻게 재실행하는가?
   반대순서로 리력목록을 어떻게 출력하는가?
7. vi편집기를 리용하기 위해 대화형편집을 어떻게 설정하는가? 이 설정을 어느

초기화파일에서 하는가?

8. fc지령은 무슨 지령인가?
9. Readline서고의 목적은 무엇인가? 어떤 초기화파일로부터 지령들을 읽는가?
10. 건맺기란 무엇인가? 무슨 건이 맺어지였는가를 어떻게 알수 있는가?
11. 종합인수란 무엇인가?
12. 다음지령들을 위한 별명들을 만드시오.

 ㄱ. clear
 ㄴ. fc-s
 ㄷ. %s - - clearb = tty
 ㄹ. kill –1

련습문제 46: 쉘메타문자

1. wildcard라는 등록부를 만드시오.
 그 등록부로 등록부를 변경시켜 재촉문에서 다음의것을 입력하시오.
 touch ab abc a1 a2 a3 a11 a12 ba ba.1 ba.2 filex filey AbC ABC ABc2 abc
2. 다음것을 하는 지령을 작성하고 검사하시오.
 ㄱ. a로 시작되는 모든 파일들을 표시하시오.
 ㄴ. 최소한 한개 수자로 끝나는 모든 파일들을 표시하시오.
 ㄷ. A 혹은 a로 시작되는 모든 파일들을 표시하시오.
 ㄹ. 2개의 알파베타문자들을 포함하는 모든 파일들을 표시하시오.
 ㅁ. 수자로 끝나는 모든 파일들을 표시하시오.
 ㅂ. 3개의 큰 문자로 이루어 진 파일들을 표시하시오.
 ㅅ. 10, 11 혹은 12로 끝나는 파일들을 표시하시오.
 ㅇ. x 혹은 y로 끝나는 모든 파일들을 표시하시오.
 ㅈ. 수자, 소문자 혹은 대문자로 끝나는 모든 파일들을 표시하시오.
 ㅊ. B 혹은 b로 시작하지 않는 모든 파일들을 표시하시오.
 ㅋ. A 혹은 a로 시작하는 2개 문자로 이루어 진 파일들을 삭제하시오.

련습문제 47: 방향바꾸기

1. 말단에 대응되는 3개의 파일의 이름은 무엇인가?
2. 파일서술자란 무엇인가?
3. 다음의 처리를 하기 위하여 어떤 지령을 사용하는가?
 ㄱ. lsfile파일을 ls지령의 출력으로 방향바꾸기하시오.
 ㄴ. lsfile파일에 date지령의 출력을 방향바꾸기하고 첨가하시오.
 ㄷ. lsfile에 who지령의 출력을 방향바꾸기하시오. 어떻게 되는가?

4. cp만을 입력하면 무슨 일이 일어 나는가?
5. 우의 실례에서 생기는 오유통보문을 파일에 어떻게 기억하는가?
6. 어미등록부로부터 시작하는 모든 파일들을 찾기 위해 find지령을 사용하시오. found파일에 표준출력을 기억하고 found.errg파일에 모든 오유들을 기억하시오.
7. 3개의 지령들의 출력을 취하여 gottem_all파일에 그 출력을 방향바꾸기하시오.
8. 얼마나 많은 프로쎄스들을 집행하고 있는가를 알기 위하여 ps와 wc지령과 함께 파이프를 사용하시오.

련습문제 48: 변수

1. 위치파라메터란 무엇인가?
   지령행에 건입력하시오.

   set dogs cats birds fish

   ㄱ. 모든 위치파라메터들을 어떻게 표시하는가?
   ㄴ. 어느 위치파라메터에 birds가 대입되는가?
   ㄷ. 위치파라메터의 번호를 어떻게 출력하는가?
   ㄹ. 쉘기억기로부터 어떻게 모든 위치파라메터들을 지우기하는가?
2. 환경변수란 무엇인가?
   그것들을 표시하려면 무슨 지령을 사용해야 하는가?
   CITY라는 환경변수를 만들고 그것에 고향이름을 대입하시오.
   그것을 어떻게 얻는가?
3. 국부변수란 무엇인가?
   사용자이름을 국부변수에 설정하시오.
   그 값을 인쇄하시오. 그것을 해제하시오.
4. declare -i의 기능은 무엇인가?
5. $$변수는 무엇을 표시하는가?
   $!변수는 무엇을 표시하는가?

제 12 장. bash쉘에 의한 프로그람작성법

제 1 절. 소개

지령들이 지령행대신에 파일로부터 실행될 때 파일을 쉘스크립트라고 하며 쉘은 비대화형으로 실행된다. bash쉘이 비대화형으로 동작할 때 환경변수 BASH_ENV(ENV)를 조사하여 그의 값으로 대입된 파일(보통 .bashrc)을 기동시킨다.

- BASH_ENV파일을 읽은 다음 쉘은 스크립트안에 있는 지령들을 실행한다.[1]

1. 쉘스크립트를 창조하는 단계

쉘스크립트는 보통 편집기에서 작성하며 설명문들이 섞인 지령들로 이루어 졌다. 설명문은 #기호로 시작되며 진행되고 있는 작업에 대하여 서술한 본문으로 구성된다.

첫번째 행 스크립트의 제일 웃왼쪽 구석의 행은 스크립트에 있는 행들을 실행하는 프로그람을 가리킨다. 보통 shbang행이라고 하는 이 행은 다음과 같이 쓴다.

```
#!/bin/bash
```

$!를 특수한 번호라고 하는데 핵심부가 스크립트안의 행들을 해석하는 프로그람을 식별하기 위해 사용된다. 이 행은 스크립트의 제일 웃행에 놓여야 한다. bash프로그람은 또한 속성을 변경시키기 위한 인수들을 쓸수 있다. bash추가선택목록은 표 12-8에 보여준다.

설명문 설명문들은 #기호로 지적된 행들이며 그자체만으로 한 행을 구성할수도 있고 스크립트지령뒤에 놓을수도 있다. 설명문들은 스크립트들을 문서화하기 위해 사용된다. 설명문을 주지 않으면 스크립트가 도대체 무엇을 하는지 리해하기가 어렵다. 설명문이 중요하지만 종종 너무 드물게 또는 거의 사용하지 않을 때가 많다. 다른 사람들을 위해서뿐아니라 자기 자신을 위해서도 무엇을 하는것인지 설명문을 주어야 한다.

실행명령문과 bash쉘구조 bash쉘프로그람은 UNIX지령들의 조합, bash쉘지령, 프로그람작성구성체와 설명문들로 이루어 졌다.

실행가능한 스크립트를 만들기 파일을 작성할 때 실행허가를 주지 않는다. 스크립트를 실행시키자면 이 허가가 필요하다. 실행허가를 얻기 위해 chmod지령을 사용한다.

실례 12-1

```
1.  $ chmod +x myscript
2.  $ ls –lF myscript
    -rwxr-xr-x   1 ellie   0 Jul   13:00 myscript*
```

[1] bash가 대화형으로 기동될 때 -norc 혹은 - -norc선택항목을 주면 BASH_ENV 혹은 ENV파일은 읽어 지지 않는다.

설명

1. chmod지령은 사용자, 그룹 기타 등에 대한 실행허가를 대입하기 위해 사용된다.
2. ls지령의 출력은 모든 사용자들이 my script파일에 대한 실행허가를 가진다는것을 가리킨다. 파일이름 끝에 있는 *도 이것이 실행가능한 프로그람이라는것을 가리킨다.

대화조종의 스크립트화 다음의 실례에서 사용자는 편집기를 사용하여 스크립트를 작성한다. 사용자가 파일을 기억한 다음 실행허가가 허용되여 스크립트가 실행된다. 프로그람에 오유가 있으면 쉘은 즉시 응답한다.

실례 12-2

```
(The Script)
1.  #!/bin/bash
2.  #This is the first Bash shell program of the day.
    #Scriptname:greetings
    #Written by:Barbara Bashful
3.  echo "Hello $LOGNAME,it's nice talking to you."
4.  echo "Your present working directory is 'pwd'."
    echo "You are working on a machine called 'uname –n'."
    echo "Here is a list of your files."
5.  ls    # List files in the present working directory
6.  echo "Bye for now $LOGNAME.The time is 'data +&T'.!"

(The Command Line)
    $ greetings      # Don't forget to turn on x permission!
    bash: ./greetings: Permission denied.

    $ chmod +x greetings
    $ greetings or   ./greetings
3.  Hello barbara, it 's nice talking to you.
4.  Your present working directory is /home/lion/barbara/prog
    You are working on a machine called lion.
    Here is a list of your files.

5.  Afile            cplus        letter       prac
    Answerbook       cprog        library      prac1
    bourne           joke         notes        per15
6.  Bye for now barbara. The time is 18:05:07!
```

설명

1. 스크립트의 첫번째 행 #!/bin/bash는 어느 쉘이 이 프로그람의 행들을 실행하는가를 핵심부에게 알려 주는데 이 경우에는 bash(Bourne Again쉘)쉘이다.
2. 설명문들은 #기호로 시작되는데 실행불가능한 행들이다. 그것들은 그자체만으로 한개 행을 이룰수도 있고 지령뒤에 첨가되여 한 행을 이룰수도 있다.

3. 쉘에 의해 변수대입이 진행된 다음 echo지령은 현시장치에 그 행을 현시한다.
4. 쉘에 의해 지령대입이 진행된 다음 echo지령은 그 행을 현시장치에 현시한다.
5. ls지령이 실행된다. 설명문은 쉘에 의해 무시된다.
6. echo지령은 2중인용부호안에 있는 문자렬을 현시한다. 변수들과 지령바꿔넣기(거꿀인용부호)들은 2중인용부호안에 있을 때 확장된다. 이 경우에는 실제상 불필요하다.

제 2 절. 사용자입력의 읽기

1. 변수(개괄)

제일 마지막장에서 변수의 선언과 해제에 대하여 설명한다. 변수들은 현재의 쉘에 의해 국부변수 또는 환경변수로 대입된다. 쉘스크립트가 다른 스크립트를 요구하지 않는 한 변수는 일반적으로 스크립트안에서 국부변수로 대입된다. 변수로부터 값을 반출하자면 변수앞에 $기호를 달아 주어야 한다. 변수는 2중인용부호안에 놓을수 있으며 쉘은 $기호를 변수확장으로 해석한다. 변수를 단일인용부호안에 넣으면 변수확장이 되지 않는다.

실례 12-3

1. name="John Doe" *or* declare name="John Doe"   *# local variable*
2. export NAME="John Doe"   *# global variable*
   # extract the value

2. read지령

read지령은 말단 또는 파일로부터 입력을 읽기 위해 사용하는 내부지령이다(표 12 - 1). read지령은 행바꾸기가 있기전까지 계속 입력한다. 행끝에 있는 행바꾸기문자를 읽을 때 빈 바이트로 번역된다. 이름이 없으면 행읽기를 내부변수 REPLY에 대입한다. read지령은 또한 행바꾸기건을 누를 때까지 프로그람을 정지시키기 위해서도 사용할수 있다. 파일로부터 행들을 읽을 때 read지령을 어떻게 사용하는가에 대해서는 691페지의 《순환지령》에서 보여 준다. read지령에 –r추가선택을 주면 거꿀빗선/행바꾸기 쌍이 무시된다. 거꿀빗선은 행의 한 부분으로 처리된다. read지령은 특성을 조종하기 위한 4개의 추가선택 즉 –a, -e, -p, -r를 가진다.[2]

표 12-1. read지령

형 식	의 미
read answer	표준입력으로부터 행을 읽고 그것을 변수 answer에 대입한다.
read first,last	표준입력으로부터 첫 공백 또는 행바꾸기전 한 행을 읽고 입력한 첫 단어를 변수 first에 넣어 그 행의 나머지부분을 변수 last에 대입한다.

[2] 선택항목 -a, -e 그리고 -p는 bash 판본 2.x에서 쓸수 있다.

(표계속)

형 식	의 미
Read	표준입력으로부터 한 행을 읽고 그것을 내부변수 REPLY에 대입한다.
read –a arrayname	단어들의 목록을 array name이라고 하는 배렬에 읽어 넣는다.ㄱ
read –e	대화형쉘에서 지령편집을 효과적으로 쓰기 위해 사용한다. 실례로 편집기가 vi이면 vi지령들을 입력행에서 사용할수 있다. ㄱ
read –p prompt	재촉문을 인쇄하고 입력을 대기하다가 들어 온 입력을 REPLY 변수에 기억시킨다.ㄱ
read –r line	입력에 거꿀빗선을 포함해도 된다. ㄱ

ㄱ. bash 판본 2.0우에서 실행되지 않는다.

실례 12-4

(The Script)

```
   #!/bin/bash
   # Scriptname: nosy
   echo –e "Are you happy? \ c"
1. read answer
echo "$answer is the right response."
echo –e "What is your full name? \ c"
2. read first middle last
   echo "Hello $first"
   echo –n "Where do you work?"
3. read
4. echo I guess $REPLY keeps you busy!
- - - - - - - - - - - - - - - - - - - - - - - - - - - -ㄴ
5. read –p "Enter your job title:  "
6. echo "I thought you might be an $REPLY."
7. echo –n "Who are your best friends?"
8. read –a friends
9. echo "Say hi to ${friends[2]}."
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

(The Output)

```
$ nosy
Are you happy? Yes
1. Yes is the right response.
2. What is your full name? Jon Jake Jones
   Hello Jon
3. Where do you work? The Chico Nut Factory
4. I guess the Chico Nut Factory keeps you busy!
5. Enter your job title: Accountant
```

6. *I thought you might be an Accountment.*
7,8. *Who are your best friends?* ***Melvin Tim Ernesto***
9. *Say hi to Ernesto*

설명

1. read지령은 사용자가 입력한 행을 접수하여 변수 answer에 대입한다.
2. read지령은 사용자로부터 입력을 접수하여 입력의 첫 단어를 변수 first에, 두번째 단어를 변수 middle에, 단어의 나머지부분을 변수 last에 대입한다.
3. 표준입력으로부터 한 행을 읽고 내부변수 last에 대입한다.
4. REPLY변수의 값을 출력한다.
5. –p추가선택에 의해 read지령은 재촉문 《Enter your job title:》를 현시하고 입력행을 특수한 내부변수 REPLY에 기억시킨다.
6. REPLY의 값을 문자렬로 현시한다.
7. 사용자에게 입력을 요구한다.
8. –a추가선택에 의해 read지령은 입력을 단어들의 배렬로 작성한다. 배렬의 이름은 friends이다. 배렬안에 읽어 들인 원소들은 Melvin, Tim, Ernesto이다.
9. friends배렬의 세번째 원소 Ernesto를 출력한다. 배렬의 첨수는 0으로부터 시작한다.

ㄴ. 이 행의 아래에 현시된 지령들은 bash 판본 2.x우에서 실행되지 않는다.

실례 12-5

(The Script)
```
    #!/bin/bash
    # Scriptname:printer_check
    # Script to clear a hung-up printer
1.  if [ $LOGNAME !=root ]
    then
        echo "Must have root privileges to run this program"
    fi
2.  cat << EOF
    Warning: All jobs in the printer queue will be removed.
    Please turn off the printer now. Press return when you are ready
    to continue.   Otherwise press Control C.
      EOF
3.  read JUNK       #Wait until the user turns off the printer
    echo
4.  /etc/rc.d/init.d/lpd stop         #Stop the printer
5.  echo –e "\ nPlease turn the printer on now."
6.  echo "Press Enter to continue"
7.  read JUNK            #Stall until the user turns the printer
                         #back on
    echo                 #A blank line is printed
8.  /etc/rc.d/init.d/lpd start     #Start the printer
```

설명

1. 사용자가 root이면 검사하고 아니면 오유를 내보내고 끌낸다.
2. here문서를 작성한다. 경고통보문을 현시장치에 현시한다.
3. read지령은 사용자입력을 대기한다. 사용자가 Enter건을 누르면 변수 JUNK는 입력한 모든것을 접수한다. 변수는 어디에도 사용되지 않는다. 이 경우 read는 사용자가 현시장치를 끄고 되돌아 가 Enter건을 누를 때까지 대기하기 위해 사용된다.
4. lpd프로그람은 인쇄기 daemon을 정지시킨다.
5. 현재인쇄기를 다시 투입할 시간이 되였다.
6. 준비되였으면 사용자가 Enter건을 누를것을 요구한다.
7. 사용자가 입력한 모든것을 변수 JUNK에 읽어 들이고 Enter건을 눌렀을 때 프로그람을 실행시킨다.
8. lpd프로그람은 인쇄기 daemon을 기동시킨다.

제 3 절. 산수

1. 옹근수(declare와 let지령)

declare지령 declare –i지령으로 변수들을 옹근수로 선택할수 있다.

사용자가 변수에 어떤 문자렬값을 대입한다면 bash는 그 변수에 0을 대입한다. 산수연산은 옹근수로 선언된 변수들의 연산에 사용된다(변수가 옹근수로 선언되지 않았다면 내부지령 let는 산수연산을 허락한다(664페지의 《let지령》에서 보여 준다.)).

만일 류동소수점을 대입하려고 하면 bash는 문법오유를 통지한다. 수자는 2진, 8진, 16진과 같이 여러가지 수체계로 표현될수 있다.

실례 12-6

1. $ **declare –i num**
2. $ **num=hello**

   $ echo $num
   0
3. $ **num=5 + 5**
   bash: +: command not found
4. $ **num=5+5**
   $ echo $num *10*
5. $ **num=4*6**
   $ echo $num
   24
6. $ **num="4 * 6"**
   $ echo $num
   24

7. $ **num=6.5**
   bash: num: 6.5: sytax error in expression (remainder of expression is ".5")

설명

1. –i추가선택이 있는 declare지령은 옹근수형변수 num을 작성한다.
2. 문자렬 hello를 옹근수변수 num에 대입하려고 한다면 그 문자렬은 0으로 기억된다.
3. let지령을 사용하지 않는한 공백을 인용부호안에 넣거나 삭제하여야 한다.
4. 공백이 소거되고 산수연산을 수행한다.
5. 곱하기연산을 수행하고 결과를 num에 대입한다.
6. 공백을 인용부호안에 넣었으므로 곱하기는 수행될수 있고 쉘은 *를 확장하지 않는다.
7. 변수를 옹근수로 대입하였기때문에 분수더하기는 bash에 의해 문법오유로 처리된다.

옹근수목록제시 –i변수만이 있는 declare지령은 이미 대입된 모든 옹근수들과 그의 값들을 현시한다. 그의 실례는 다음과 같다.

```
 $ declare –i
declare –ir EUID=”15”            # effective user id
declare –ir PPID=”235”           # parent process id
declare –ir UID=”15”             # user id
```

여러가지 수체계의 표현과 사용 수들을 10진수(기정으로 기초수는 10이다.), 8진수(기초수 8), 16진수(기초수 16)로 표현할수 있다.

형식

variable=base#number-in-that-base

실례 12-7

N=2#101 *Base is 2; number 101 is in base 2*

실례 12-8

(The Command Line)

1. $ **declare –i x=017**
   $ echo $x
   15
2. $ **x=2#101**
   $ echo $x
   5
3. $ **x=8#17**

```
$ echo $x
15
```

4. $ **x=16#b**

```
$ echo $x
11
```

설명

1. declare함수는 옹근수형변수 x에 8진수 값 017을 대입하기 위해 사용된다. 8진수는 0부터 시작해야 한다. 8진수 값 017을 10진수 15로 인쇄한다.
2. 변수 x에 101(2진수)을 대입한다. 여기에서 2는 #로 구별되는 기초수체계를 그리고 기초수체계에서의 수는 101이라는것을 나타낸다. x의 값을 10진수 5로 인쇄한다.
3. 변수 x에 8진수 17을 대입한다. x의 값을 10진수값 15로 인쇄한다.
4. 변수 x에 16진수 b를 대입한다. x의 값은 10진수 11로 인쇄된다.

let지령 let지령은 옹근수연산과 수값검사를 하는 bash쉘내부지령이다. bash판본에 제공되는 let추가선택들을 보려면 재촉문이 나타났을 때 다음과 같이 입력한다.

help let

실례 12-9

1. $ **i=5** or **let i=5**
2. $ **let i=i+1**
   $ **echo $i**
   6
3. $ **let "i= i + 2"**
   $ **echo $i**
   8
4. $ **let "i+=1"**
   $ **echo $i**
   9
5. $ **i=3**
6. $ **((i+=4))**
   $ **echo $i**
   7
7. $ **((i=i-2))**
   $ **echo $i**
   5

설명

1. 변수 i에 값 5를 대입한다.
2. let지령은 i값에 1을 더한다. $는 산수연산을 진행할 때 변수바꿔넣기를 요구하지 않는다.
3. 인용부호는 인수가 공백을 포함하고 있을 때 필요하다.

4. 지름연산자 +=는 값에 1을 더하기 위해 사용한다.
5. 변수 i에 값 5를 대입한다.
6. 2중인용부호는 let를 바꿔넣기하는데 사용할수 있다.[ㄱ]
   4가 더해 지고 i에 대입된다.
7. i에서 2를 던다. i - =2로도 쓸수 있다.

2. 류동소수점산수연산

bash는 옹근수연산만 제공하지만 bc, awk 그리고 newk편의프로그람들은 복소수연산까지도 실행할수 있다.

실례 12-10

(The Command Line)

```
1. $ n='echo "scale=3; 13 / 2" | bc'
   $ echo $n
   6.500
2. product='gawk –v x=2.45   –v y=3.123 'BEGIN{printf "%.2f\ n      ",x*y}' '
   $ echo $product
   7.65
```

설명

1. echo지령의 출력은 bc프로그람에로 파이프된다. 수자표기법은 소수점아래 3자리까지 현시한다. 계산값은 2로 13을 나눈것이다. 파이프행은 인용부호로 처리된다. 지령바꿔넣기가 실행되여 출력은 변수 n에 대입된다.
2. gawk프로그람은 지령행에서 x=2.45 y=3.123으로 넘겨 진 인수목록으로부터 그 값을 얻는다. 수들을 곱한 다음 printf함수는 소수점아래 2자리의 정확도를 가진 결과를 형식화하여 인쇄한다. 출력을 변수 product에 대입한다.

제 4 절. 위치파라메터와 지령행인수

1. 위치파라메터

정보는 지령행을 통해 스크립트안을 통과할수 있다. 스크립트다음에 있는 매 단어(공백으로 구별되여 있다.)들을 인수라고 한다. 지령행인수는 위치파라메터로 스크립트안에서 참고될수 있다. 실례로 첫번째 인수는 $1, 두번째 인수는 $2, 세번째 인수는 $3등으로 처리된다. $9다음의 대괄호는 한개 수로서 그수를 보존하기 위해 사용된다. 실례로 위치파라메터10은 ${10}으로 참조한다. $# 변수는 파라메터들의 수를 검사하기 위하여 사용하며 $*는 그 모든것을 현시하기 위해 사용한다. 위치파라메터는 set지령에 의해 설정 또는 재설정할수 있다. set지령을 사용할 때 임의의 위치파라메터들은 원래의 모임을 지워 버린다. 표 12 - 2에서 보여 준다.

[ㄱ]. 이중인용부호《 》는 bash 2.x 판본에서 let 를 바꿔넣기 위해 사용된다.

표 12-2.　위치파라메터

위치파라메터	무엇을 참조하는가?
$0	스크립트의 이름을 참조한다.
$#	위치파라메터의 번호값을 유지한다.
$*	모든 위치파라메터들을 표시한다.
$@	2중인용부호안에 있을 때를 제외하고는 $*와 같다.
"$*"	단일인수로 확장한다("$1, $2, $3"와 같다.).
"$@"	인수를 분리하여 확장한다("$1", "$2", "$3"과 같다.).
$1 $[10]	개별적인 위치파라메터들을 참조한다.

실례 12-11

(The Script)
```
     #!/bin/bash
     #Scriptname:greetings2
     echo "This script is called $0."
1.   echo "$0   $1 and $2"
     echo "The number of positional parameters is $#"
  ---------------------------

(The Command Line)
     $ chmod +x greetings2
2.   $ greetings2
     This script is called greetings2.
     greetings and
     The number of positional paramters is
3.   $ greetings2 Tommy
     This script is called greetings2.
     greetings Tommy and
     The number of positional parameters is 1
4.   $ greeting2 tommy Kimberly
     This script is called greetings2
     greetings Tommy and Kimberly
     The number of positional parameters is 2
```

설명

1. 스크립트 greeting2에서 위치파라메터 $0은 스크립트이름을 참고하며 $1은 첫번째 지령행일치 그리고 $2는 두번째 지령행의 일치를 참고한다.
2. greetings2스크립트는 아무런 인수들을 넘겨줌이 없이 실행한다. 출력은 greetings라는 스크립트와 $1과 $2에 아무것도 대입되지 않았다는것을 설명한다. 그러므로 그 값들은 빈값이며 아무것도 인쇄되지 않는다.
3. 이때 한개의 인수 Tommy를 넘겨 준다. Tommy는 위치파라메터 1에 대입된다.

4. 2개의 인수 Tommy와 kimberly를 입력시킨다. Tommy를 $1에 대입하며 kimberly를 $2에 대입한다.

2. set지령과 위치파라메터

set지령에 인수를 주면 위치파라메터들이 다시 대입된다.[3] 다시 대입을 하면 낡은 파라메터목록을 읽는다. 모든 위치파라메터들을 해제하기 위해서는 set - - 를 사용해야 한다. $0은 항상 스크립트의 이름이다.

실례 12-12

(The Script)

```
    #!/bin/bash
    # Scriptname :   args
    # Script to test command line arguments
1.  echo The name of this script is $0.
2.  echo The arguments are $*.
3.  echo The first argument is $1.
4.  echo The second argument is $2.
5.  echo The number of arguments is $#.
6.  oldargs=$*
7.  set Jake Nicky Scott      # Reset the positional parameters
8.  echo All the positional parameters are $*.
9.  echo The number of postional parameters is $#.
10. echo "Good-bye for now, $1."
11. set $(date)               # Reset the positional parameters
12. echo The date is $2 $3, $6.
13. echo "The value of\ $oldargs is $oldargs."
14. set $oldargs
15. echo $1 $2 $3
```

(The Output)

```
    $ args a b c d
1.  The name of this script is args.
2.  The arguments are a b c d.
3.  The first argument is a.
4.  The second argument is b.
5.  The number of arguments Is 4.
6.  All the positional parameters are Jake Nicky Scott.
7.  The number of positional parameters is 3.
8.  Good-bye for now, Jake.
9.  The date is Mar 25, 2001.
10. The value of $oldargs is a b c d.
```

---

3. set 지령에 인수를 주지 않으면 이 쉘에서 설정된 모든 변수 즉 국부변수와 반출변수들을 현시한다. set 지령에 선택항목을 주면 x 와 v 선택항목과 같은 쉘조종선택항목들을 설정 및 해제한다.

11. *Wed Mar 25*

설명

1. 스크립트의 이름을 $0변수에 기억한다.
2. $*는 모든 위치파라메터들을 현시한다.
3. $1은 첫번째 위치파라메터를 현시한다(지령행인수).
4. $2는 두번째 위치파라메터를 현시한다(지령행인수).
5. $#는 위치파라메터들의 총계이다(지령행인수들).
6. 모든 위치파라메터들이 oldqrgs변수에 기억된다.
7. set지령은 낡은 목록들은 지우면서 위치파라메터들을 재설정하게 한다. 이때 $1은 Jake, $2는 Nicky 그리고 $3은 Scott이다.
8. $*는 모든 파라메터들 즉 Jake, Nicky 그리고 Scott를 현시한다.
9. $#는 파라메터의 수 즉 3을 나타낸다.
10. $1은 Jake이다.
11. 지령바꿔넣기를 한 다음 date를 실행하고 위치파라메터들을 date지령의 출력으로 재설정한다.
12. $2, $3 그리고 $6의 새로운 값들을 현시한다.
13. oldargs에 기억된 값들을 인쇄한다.
14. set지령은 oldargs안에 기억된 값들로부터 위치파라메터들을 만든다.
15. 첫 3개의 위치파라메터들을 현시한다.

실례 12-13

```
(The Script)
  #!/bin/bash
  # Scrlptname: checker
  # Script to demonstrate the use of special variable
  # modifiers and arguments
1. name=${l:?"requires an argument" }
   echo Hello $name

(The Command Line)
2. $ checker
   checker: 1: requires an argument
3. $ checker Sue
   Hello Sue
```

설명

1. 특수한 변수변경자 :?는 $1이 값을 가지는가 안가지는가를 검사한다. 없으면 스크립트에서 탈퇴하고 통보문을 현시한다.
2. 프로그람은 인수 없이 실행되며 $1에는 값이 대입되지 않는다. 오유가 현시된다.
3. checker프로그람은 지령행인수 sue를 준다. 스크립트에서 $1에는 sue를 대입한다. 프로그람은 계속 실행된다.

$*와 $@의 차이 $*와 $@는 오직 2중인용부호안에 있을 때만 차이난다. $*가 2중인용부호로 처리될 때 파라메터목록은 단일문자렬로 된다. $@가 2중인용부호안에 있을 때 매 파라메터들은 인용부호로 처리된다. 즉 매 단어는 구분문자렬로 처리된다.

실례 12-14

```
1. $ set 'apple pie' pears peaches
2. $ for i in $*
   > do
   > echo $i
   > done
   apple
   pie
   pears
   peaches

3. $ set 'apple pie' pears peaches
4. $ for i in "$*"
   > do
   > echo $i
   > done
   apple pie pears peaches

5. $ set 'apple pie' pears peaches
6. $ for i in $@
   > do
   > echo $i
   > done
   apple
   pie
   pears
   peaches

7. $ set 'apple pie' pears peaches
8. $ for i in "$@"            # At last!!
   > do
   > echo $i
   > done
   apple pie
   pears
   peaches
```

설명

1. 위치파라메터들이 설정된다.
2. $*가 확장될 때 apple pie를 넣은 인용부호는 해제된다. 즉 apple과 pie는 2개의 분리된 단어들로 된다. for순환은 변수 i에 매 단어들을 대입하고 i의 값을 인쇄한다. 순환하면서 i의 값을 써넣어 매 단어들을 대입한다. 순환에 의해 매 단계마다 왼쪽으로 자리밀기하며 다음 단어가 변수 i에 대입된다.

3. 위치파라메터들을 설정한다.
4. 2중인용부호안에 $*를 넣으면 입력파라메터목록은 한개의 줄에 apple pie pears peaches로 된다. 입력목록은 한개 단어씩 i에 대입된다. 순환은 한번 반복한다.
5. 위치파라메터를 설정한다.
6. 인용부호안에 넣지 않은 $@와 $*는 같은것으로 표현된다.
7. 위치파라메터를 설정한다.
8. 2중인용부호안에 $@를 넣으면 매 위치파라메터들은 한개 인용부호안에 넣은 문자렬로 처리된다. 목록은 apple, pie, pears 그리고 peaches로 된다. 요구되는 결과에 정확히 도달하였다.

제 5 절. 조건구성체와 흐름조종

1. 탈퇴상태

조건지령들은 조건이 성공인가 아니면 실패인가에 기초한 과제를 수행하게 한다. if 지령은 여기에서 제일 간단한 형태이다. 즉 if/else지령은 두가지 채택을 허용하며 if/elif/else지령은 다중조건을 허용한다. Bash는 두가지 조건형태를 검사할수 있다. 즉 지령이 성공인가 아니면 실패인가와 표현식이 참인가 아니면 거짓인가 하는것이다. 이 경우 항상 탈퇴상태가 사용된다. 탈퇴상태 0은 항상 성공이면서 참이라는것을 가리키고 령이 아닌 탈퇴상태는 실패 이면서 거짓이라는것을 가리킨다. 상태변수 ?는 탈퇴상태를 현시하는 수값을 가지고 있다. 탈퇴상태들이 어떻게 동작하여 기억기를 되살리는가는 실례 12-15를 참고하면 된다.

실례 12-15

(At the Command Line)
1. $ name=Tom
2. $ grep "$name" /etc/passwd
   Tom:8ZKX2F:5102:40:Tom Savage:/home/tom:/bin/sh
3. $ **echo $?**
   0 *# Success!*
4. $ name=Fred
5. $ **grep "$name" /etc/passwd**
   $ **echo $?**
   1 *# Failure*

설명

1. 변수 name에 문자렬 Tom을 대입한다.
2. grep지령은 passwd파일에서 문자렬을 검사한다.
3. 변수 ?는 제일 마지막지령실행에 관한 탈퇴상태를 가진다. 이 경우 grep의 탈퇴상태가 대입된다. grep지령이 문자렬 Tom을 찾으면 탈퇴상태 0으로 귀환된다. grep지령은 성공한다.

4. 변수 name에 Fred를 대입한다.
5. grep지령은 passwd파일에서 fred를 탐색하였는데 그것을 찾을수 없었다. ? 변수는 grep가 실패했다는것을 표시하는 값 1을 가진다.

2. 내부지령 test

내부지령 test를 사용하여 표현식을 평가한다. 이 지령은 또한 중괄호와 련결되여 있다. test지령 그자체를 사용할수도 있고 표현식을 한개 중괄호안에 넣을수도 있다. 쉘메타문자확장을 단순한 test의 지령으로 평가하는 식에서나 중괄호를 사용할 때는 진행하지 않는다(실례 12-16).

bash2. x판본에서는 2중중괄호[[]](내부합성지령 test)를 표현식을 평가하는데 사용할수 있다. 변수에 있는 단어는 분리하지 못하며 패턴대조가 진행되여 메타문자의 확장을 하게 된다. 공백이 포함되여 있는 자모문자렬은 인용부호안에 있어야 하며(공백 또는 공백이 없는) 문자렬을 패턴의 일부분으로가 아니라 정확한 문자렬로서 평가하려면 인용부호안에 넣어야 한다.론리연산자 &&(and)와 ||(or)는 단순한 test지령으로 사용되는 –a와 –o연산자를 교체한다(실례 12-17).

test지령이 산수연산식을 평가할수 있어도 c에서와 같은 풍부한 연산자들을(bash 2.x) 써서 사용하는것이 편리하다. let지령을 2중소괄호안에 식을 넣어서 다르게 나타낼수 있다(실례 12-18).

test지령, 합성지령, let지령을 사용하여도 검사된 식의 결과는 성공을 나타내는 상태 0과 실패를 나타내는 상태 0이 아닌 값을 가진다(표 11-10을 볼것). 다음의 실례들은 탈퇴상태가 내부지령 test, test지령의 다른형식인 단일중괄호 [];, 2중중괄호인 합성지령과 2중소괄호인 let지령에 의해 어떻게 검사되는가 하는것을 설명해 주고 있다.

실례 12-16

(The *test* Command)
(At the Command Line)

```
1.  $ name=Tom
2.  $ grep "$naa»e" /etc/paaswd
3.  $ echo $?
4.  $ test $name != Tom
5.  $ echo $?
     1                # Failure
6.  $ [ $name = Tom ]
              # Brackets replace the test command
7.  $ echo $?
     0
8.  $ [ $name = [Tt]?? ]
     $ echo $?
     1
9.  $ x=5
     $ y=20
10. $ [ $x -gt $y ]
```

```
     $ echo $?
     1
11.  $ [ $x -le $y ]
      $ echo $?
     0
```

설명

1. 변수 name에 문자렬 Tom을 대입한다.
2. grep지령은 passwd파일안에서 문자렬 Tom을 탐색한다.
3. ? 변수에는 마지막으로 실행된 지령의 탈퇴상태가 들어 있는데 이 경우에는 grep의 탈퇴상태가 들어 있다. grep가 문자렬 Tom을 찾으면 탈퇴상태 0을 돌려 준다. grep지령은 파일에서 문자렬 Tom을 찾았다.
4. test지령은 문자렬, 수자들을 평가하기 위해서와 파일시험을 하기 위해서 사용한다. 모든 지령에서처럼 이 지령도 탈퇴상태를 돌려 준다. 탈퇴상태가 령이면 표현식은 참이며 탈퇴상태가 1이면 표현식은 거짓이다.같기부호 량옆에는 공백이 반드시 있어야 한다. Name의 값이 Tom과 같지 않은가를 보기 위해 검사한다.
5. 검사가 실패하여 탈퇴상태 1을 돌려 준다.
6. 중괄호는 test지령에 대한 다른 표기법이다. 첫번째 중괄호다음에는 공백이 놓여야 한다. name이 문자렬 Tom과 같은가 하는것을 알기 위해 표현식을 검사한다. bash에서 문자렬의 동등성에 대한 검사에 단일 같기부호 또는 2중같기부호를 사용한다.
7. 검사된 탈퇴상태는 0이다. $name이 Tom과 같으면 시험은 성공하였다.
8. test지령은 통용기호확장을 허락하지 않는다. 물음표가 자모문자로 취급되였기 때문에 시험은 실패한다. Tom과 [TTt]??는 같지 않다. 8행에 있는 본문이 실패하였다는것을 나타내는 탈퇴상태는 1이다.
9. x와 y에 수값을 대입한다.
10. test지령은 수값관계연산자를 사용하여 연산수들을 검사하는데 이 실례에서는 $x가 $y보다 큰가를 검사하여 참이면 탈퇴상태 0을 돌려 주고 거짓이면 1을 돌려 준다(표 12-3에서 보여 준다.).
11. $x가 $y보다 작거나 큰가를 검사하여 참이면 탈퇴상태 0을, 거짓이면 탈퇴상태 1을 돌려 준다.

실례 12-17

(The compound *test* command)(bash 2.x)

```
   $ name=Tom; friend=Joseph
1. $ [[ $name == [Tt]om ]]          # Wildcards allowed
   $ echo $ ?

   0

2. $ [[ $name == [Tt]om && $friend == "Jose" ]]
   $ echo $?
```

```
        1
3.  $ shopt –s extglob           # Turns on extended pattern matching
4.  $ name=Tommy
5.  $ [[ $name == [Tt]o+(m)y ]]
    $ echo $?
    0
```

설명

1. test합성지령을 사용하면 쉘메타문자를 문자렬검사에 사용할수 있다. 이 실례에서 표현식은 name이 Tom, tom 또는 tommy들과 같은 문자렬중의 하나와 대조할수 있는 문자렬들로서 검사된다. 만일 표현식이 참이면 탈퇴상태(?)는 0이다.
2. 론리연산자 &&와 ||은 합성검사에 사용할수 있다. &&를 사용하면 두 표현식이 참이여야 하며 만일 첫번째 표현식이 거짓이라면 구태여 검사를 더 하지 않는다. ||론리연산자에서는 다만 한개의 표현식만이 참이면 된다. 만일 첫번째 표현식이 참이면 검사를 더 진행하지 않아도 된다. "Jose"를 인용괄호안에 넣었다. 만일 그렇게 하지 않으면 friend변수가 패턴 Jose를 포함하는가를 검사한다. Jose가 일치하는데 변수는 Joseph로 되여 있다. 표현식은 두번째 조건식이 참이 아니기때문에 탈퇴상태 1로 된다.
3. 확장된 패턴대조는 내부지령 shopt에 의해 이루어 진다.
4. 변수에 값 Tommy를 대입한다.
5. 이 검사에서 표현식은 문자렬동등성에 대한 검사를 새로운 패턴대조메타문자를 사용하여 진행한다. m이 T또는 t로 시작하고 다음에 0이 있고 그다음에 하나 또는 m개이상의 문자가 놓이며 그다음에 y가 놓이는 문자렬과 일치하는가를 검사한다.

실례 12-18

(The *let* Command) (bash 2.x)
(At the Command Line)

```
1.  $ x=2
    $ y=3
2.  (( x > 2 ))
    echo $?
    1

3.  (( x < 2 ))
    echo $?
    0

4.  (( x == 2 && y == 3 ))
    echo $?
```

0
5. ((x>2||y<3)) echo $?
echo $?
1

설명

1. x와 y에 수값을 대입한다.
2. 2중소괄호는 수값표현식을 평가하기 위해 let지령을 바꿔넣기한다. x가 y보다 크면 탈퇴상태는 0이다. 조건이 참이 아니기때문에 탈퇴상태는 1이 된다. 변수 ?는 맨 마지막으로 실행된 지령 즉 (( ))지령의 탈퇴상태를 가지게 된다. 변수를 계산할 때 화페기호($)는 변수가 (( ))에 둘러 막혀 있을 때에는 필요 없다.
3. 2중소괄호는 표현식을 평가한다. x가 2보다 작으면 탈퇴상태 0을 주고 그렇지 않으면 1을 준다.
4. 합성표현식을 평가한다. 표현식은 다음과 같이 검사된다. x가 2보다 크거나 혹은 y가 3보다 작으면 탈퇴상태는 0으로 되고 그렇지 않으면 1로 된다.

표 12-3. test지령의 연산자

Test연산자	참이면검사
문자렬검사	
[string 1=string 2]	string 1이 string 2와 같다.
[string 1==string 2]	Bash 2.x에서 =부호는 신호대신 사용된다.
[string 1!=string 2]	문자렬 1이 문자렬 2와 같지 않다.
[문자렬]	문자렬은 빈값이 아니다.
[-zstring]	문자렬의 길이가 0이다.
[-nstring]	문자렬의 길이가 0이 아니다.
[-l string]	문자렬의 길이
실례:	test –n 혹은 [-n$word]
	testtom =sue 혹은 [tom=sue]
론리검사	
[string 1 –a string 2]	문자렬 1과 문자렬 2가 둘다 참이다.
[string 1 -0 string 2]	문자렬 1과 문자렬 2의 둘중 하나가 참이다.
[!string 1]	문자렬 1이 아니다.
론리검사(복합검사)ㄱ	
[[pattern1 && pattern2]]	패턴 1과 패턴 2가 둘다 참이다.

(표계속)

Test연산자	참이면 검사
[[pattern1 || pattern2]]	패턴1과 패턴2의 둘중 하나가 참이다.
[[!pattern]]	패턴이 아니다.
옹근수검사	
[integer1 –eq integer2]	옹근수 1이 옹근수 2와 같다.
[integer1 -ne integer2]	옹근수 1이 옹근수 2와 같지 않다.
[integer1 –gt integer2]	옹근수 1이 옹근수 2보다 크다.
[integer1 –geinteger2]	옹근수 1이 옹근수 2와 같거나 크다.
[integer1 –lt integer2]	옹근수 1이 옹근수 2보다 작다.
[integer1 –le integer2]	옹근수 1이 옹근수 2보다 작거나 같다.
파일시험을 위한 2진연산	
[file 1 – nt file 2]	만일 파일 1이 파일 2보다 새것이면 참이다.
[file 1 – ot file 2]	만일 파일 1이 파일 2보다 낡은것이면 참이다.
[file 1 – ef file 2]	만일 파일 1과 파일 2가 같은 장치이거나 또는 수값을 가지면 참이다.

ㄱ. 복합검사에서 패턴은 메타문자들과 일치하는 패턴을 포함할수 있다. 문자렬검사를 정확히 하기 위해 pattern2를 인용부호안에 넣어야 한다.

표 12-4. let지령연산자

연산자	의 미
- +	덜기와 더하기
!~	부정
* / %	곱하기, 나누기, 나머지
+ -	더하기, 덜기
let연산자는 bash 2.x보다 먼저 실행되지 말아야 한다.	
<< >>	왼쪽 밀기, 오른쪽 밀기
<= >= < >	비교연산자들
==, !=	같거나 같지 않기
&	적
ˆ	안맞음합
		론리적
||	론리합

(표계속)

연산자	의 미
== /= %=	대입과 지름대입
+= -= <<= >>=	
&= ^= |=	

3. if지령

가장 간단한 조건형식은 if지령이다. if다음의 지령(bash지령이나 혹은 실행가능한 지령)이 실행되여 탈퇴상태가 되돌아 온다. 탈퇴상태는 보통 편의프로그람을 작성하는 프로그람작성자에 의해 결정된다. 탈퇴상태가 령이면 지령은 성공을 의미하며 then예약어다음의 명령문들이 실행된다. C쉘에서 if지령다음의 표현식은 C에서와 같은 론리형태의 표현식이다. 하지만 bash와 Bourne 그리고 Korn쉘에서 if다음에 놓이는 명령문은 지령이나 혹은 지령묶음이다. 만약 평가되는 지령의 탈퇴상태가 령이면 then다음의 명령블로크들은 fi가 나타날 때까지 실행된다. fi는 if블로크의 끝을 나타낸다. 탈퇴상태가 령이 아니면 지령이 실패라는것을 의미하며 then다음의 명령문들이 무시되고 조종은 fi명령문다음에 있는 행으로 직접 넘어 간다. 검사되는 명령문의 탈퇴상태를 아는것이 중요하다. 실례로 grep지령에서 탈퇴상태는 grep지령이 파일에서 패턴을 찾았는가 또는 못찾았는가를 알려고 할 때 사용할수 있다.

grep가 검색에서 성공하면 탈퇴상태 0을 돌려 주고 그렇지 않으면 령이 아닌 값을 되돌린다. sed와 gawk프로그람도 역시 패턴을 검색하지만 이것들은 패턴을 찾았는가 못찾았는가 하는데서 성공한 탈퇴상태만을 알려 준다.

형식

```
if 지령
then
        지령
      지령
fi
-----------------------------------
      (수들과 문자렬들을 위한 test사용 -- 낡은  형식)
          if test 표현식
          then
               지령
          fi
                 or
          if [문자렬/수값표현식] then
               지령
          fi
-----------------------------------
     (문자렬을 위한 test사용 ----새로운  형식)
 if [[문자렬  표현식]]then
 지령
```

```
        fi
    (수들을 위한 let사용 ---새로운  형식)
  if ((수값표현식))
```

실례 12-19

```
1. if grep "$name" /etc/passwd > /dev/null 2>&l
2. then
    echo Found $name!
3. fi
```

설명

1. grep지령은 /etc/passwd자료기지에서 인수 `name을 검색한다. 표준출력과 표준오유를 /dev/null로 방향바꾸기한다(UNIX비트바키트).
2. grep지령의 탈퇴상태가 령이면 프로그람은 then명령문으로 가며 fi가 나타날 때까지 지령들을 실행한다. then과 fi에 약어사이에 있는 지령들은 프로그람을 보다 쉽게 읽게 하려고 할 때 사용되기때문에 오유수정이 쉽다.
3. fi는 then명령문다음의 지령들의 목록을 끝맺는다.

실례 12-20

```
1. echo   "Are you o.k. (y/n) ?"
    read answer
2. if [ $answer   = y -o "$answer" = y ]
    then
    echo   "Glad to hear it."
3. fi
4. if [ $answer = Y -o "$answer" = y ]
    [: too many arguments

5. if [[ $answer = = [Yyl* | | $answer = =   Maybe ]] ᄀ
    then
    echo "Glad to hear it."
    fi

6. shopt -s extglob

7. answer="not really"

8. if [[ $answer = [Nn]o?( way|t really) ]]
    then
        echo "So sorry. "
    fi
```

설명

1. 사용자에게 질문을 제기하고 응답을 요구한다. read지령은 응답을 기다린다.
2. 중괄호로 나타내는 test지령은 표현식을 검사하기 위해 사용된다. 표현식이 참이면 탈퇴상태값 0으로, 표현식이 거짓이면 령이 아닌 값을 되돌린다. 변수 answer가 Y 또는 y이면 then다음의 지령들을 실행한다(test지령은 검사표현식에 *를 사용하지 않는다. 그리고 =연산자들처럼 공백이 중괄호량옆에 있어야 한다. 표 12-3에서 보여 준다.). $ answer는 한개 문자렬로 취급하기 위해 2중인용부호안에 넣는다. 그렇지 않으면 test지령이 실패하게 된다.
3. fi는 2행에 있는 if를 끝낸다.
4. test지령은 =연산자앞에 한개이상의 단어가 있으면 실패하게 된다. 실례로 사용자가 yes, you betcha를 넣으면 answer변수는 3개 단어로 평가하기때문에 $answer가 2중인용부호안에 있지 않으면 test는 오유통보문을 여기서 보여 준다.
5. 합성연산자 [[ ]]는 문자렬표현식에서 쉘메타문자표현식을 허용한다. 이전의 test지령에서는 변수가 여러개의 단어들을 포함하고 있어도 그것을 인용부호안에 넣을 필요가 없었다.
6. 만일 extglob를 설정하면 shopt내부지령은 파라메터표현식확장을 허용한다. 표 11-12에서 보여 준다.
7. answer변수는 문자렬 "not really"로 설정된다.
8. 확장된 패턴대조가 여기에서 사용된다. 만약 answer변수의 값이 no 혹은 No로 시작하고 다음에 괄호안에 있는 표현식가운데서 빈값 혹은 한개로 이루어 지면 표현식은 참으로 된다. 표현식을 no, No, no way, No way, not really, Not really로 평가할수 있다.

ㄱ. 행 5부터 8까지는 bash 판본 2.x에서만 실행된다.

exit지령과 ?변수 exit지령은 스크립트를 끝내거나 지령행으로 되돌려 보내는데 사용된다. 어떤 조건이 주어 지면 스크립트를 탈퇴하려고 한다. exit지령에 주어 진 인수는 0으로부터 255까지의 범위에 있는 수이다. 프로그람이 인수 0으로 탈퇴하면 성공적으로 탈퇴한다는것을 의미한다. 령이 아닌 인수는 실패의 종류를 나타낸다. exit지령에 주어진 인수는 쉘의 ?변수에 기억된다.

실례 12-21

```
(The Script)
 $ cat bigfiles
    # Name: bigfiles
    # Purpose: Use the find command to find any files in the root
    # partition that have not been modified within the past n (any
    # number within 30 days) days and are larger than 20 blocks
    # (512-byte blocks)
1.  if (( $#!= 2 ))ㄱ              #[ $# -ne 2 ]
```

```
            then
              echo    "Usage:     $0 mdays size " 1>&2
              exit 1
    2.  fi
    3.  if (( $1 <   0 |  | $1 > 30 ))ㄴ        #[ $1 -It 0 -o $1 -gt 30 ]
            then
              echo "mdays is out of range"
              exit 2
    4.  fi
    5.  if (( $2 <= 20 ))         #[ $2 -le 20 ]
            then
              echo "size is out of range"
              exit 3
    6.   fi
    7.  find / -xdev –mtime $1 -size +$2
    (The Command Line)
        $ bigfiles
        Usage: bigfiles mdays size
        $ echo $?
        1
        $ bigfiles 400 80
        mdays is out of range
        $ echo $?
        2
        $ bigfiles 25 2
        size is out of range
        $ echo $?
        3
        $ bigfiles 2 25
    (Output of find prints here)
```

설명

1. 명령문을 다음과 같이 읽는다. 인수의 개수가 2가 아니면 오유통보문을 인쇄하고 표준오유에 보낸 다음 탈퇴상태 1로 스크립트를 탈퇴한다. 내부지령 test 혹은 let지령들은 수값표현식을 검사하는데 사용할수 있다.
2. fi는 then다음명령문의 블로크끝을 나타낸다.
3. 명령문을 다음과 같이 읽는다. 지령행으로부터 넘어 온 첫번째 위치파라메

터의 값이 0보다 작거나 30보다 크면 통보문을 인쇄하고 탈퇴상태 2로 탈퇴한다. 수값연산자에 대해서는 표 12-4에서 보여 준다.

4. fi는 if블로크를 끝낸다.
5. 명령문을 읽는다. 지령행에서 넘어 온 두번째 파라메터의 값이 20(512byte 블로크)보다 작거나 같으면 통보문을 인쇄하고 탈퇴상태 3으로 탈퇴한다.
6. fi는 if블로크를 끝낸다.
7. find지령은 뿌리등록부에서 검색을 시작한다.
   –xdev추가선택은 find지령이 다른 구획을 검색하지 않게 한다.
   -mtime추가선택은 파일이 변경되였을 때의 날자값인 수값인수를 가진다.
   -size추가선택은 512byte블로크로 구성되는 파일의 크기인 수값인수를 가진다.

ㄱ. bash판본 2.x에서는 실행되지 않는다. 낡은 판본에서는 if let $(($# !=2))로 쓴다.
ㄴ. 우와 같다. 낡은 판본에서는 if let $(($1 < 0 ‖ $1 > 30)로 쓴다.

NULL값검사 변수가 빈값을 가지는가를 검사할 때 빈값인가 혹은 test지령의 실패인가 하는것을 보여 주기 위해 2중인용부호를 사용한다.

실례 12-22

(The Script)

```
1.  if [ "$name" = "" ]          # Alternative to  [  ! " $name" ]
                                         # or  [  -z "$name" ]
    then
            echo The name variable is null
    fi
(From System showmount program, which displays all remotely mounted systems)
remotes=$(/usr/sbin/showmount)
2   if [ "X${remotes}"  ! = "X" ]
            then
            /usr/sbin/wall ${remotes}
3.  fi
```

설명

1. name변수가 빈값이면 test지령은 참이다. 2중인용부호는 빈값을 나타내기 위해 사용된다.
2. showmount지령은 주콤퓨터로부터 원격으로 탑재된 모든 의뢰기들을 나타낸다. 지령은 한개이상의 의뢰기를 나타내거나 혹은 아무것도 나타내지 않을수도 있다. 변수 remotes는 대입된 값 아니면 빈값을 가진다. 검사할 때 문자 X를 변수 remotes앞에 놓아야 한다. remotes가 빈값이면 의뢰기들이 원격으로 등록가입된것들이 없으며 x는 3행에 의해 다시 프로그람이 실행되기때문에 x와 같다. 변수가 값 실례로 주이름 pluto을 가지면 표현식 if xpluto!=x를 읽고 wall지령을 실행한다(모든 사용자들에게 통보문을 보낸다.). 표현식에서 x를 사용하는 목적은 remotes의 값이 빈값이라는 담보를 주자는것이므로 X는 항상 표현식에서 !=연산자의 량쪽에 배치된다.
3. fi는 if를 끝낸다.

겹쌓인 if지령 if명령문들이 겹쳐 졌으면 fi명령문은 항상 제일 가까운 if명령문에로 간다. 겹ifs를 쑥 들여다 쓰기하는것은 어느 if명령문이 어느 fi명령문으로 가는가를 쉽게 보기 위한것이다.

4. if/else지령

if/else지령들은 2개의 채택처리를 하게 한다. If다음의 지령이 거짓이면 else다음의 지령이 실행된다.

형식

```
f지령
then 지령(들)
else 지령(들)
fi
```

실례 12-23

```
(The Script)
    #!/bin/bash
    # Scriptname: grepit
1.  if grep "$name" /etc/passwd >•& /dev/null; then
2.  echo Found $name!
3.  else
4.          echo "Can't find $nanie."
            exit 1
5.   fi
```

설명

1. grep지령은 NIS passwd자료기지에서 인수 name에 대하여 검색한다. 사용자가 출력을 볼 필요가 없기때문에 표준출력과 표준오유를 /dev/null로 방향바꾸기한다(UNIX비트바키트).
2. ypmatch지령의 탈퇴상태가 0이면 프로그람조종은 then명령문으로 가며 else가 나타날 때까지 지령들을 실행한다.
3. else명령문다음에 있는 지령들은 ypmatch지령이 passwd자료기지안에서 $name탐색을 실패하면 실행된다. 즉 else블로크안에 있는 지령들을 실행하기 위해서는 ypmatch의 탈퇴상태가 0이 되지 말아야 한다.
4. $name의 값을 passwd자료기지에서 찾지 못하면 echo명령문이 실행되며 프로그람은 실패를 나타내는 탈퇴상태값 1을 가진다. fi는 if를 끝낸다.

실례 12-24

```
(The Script)
    #!/bin/bash
    # Script-name: idcheck
```

```
    # purpose:check user id to see if user is root.
    # Only root has a uid of 0.
    # Format for id output:uid=9496(ellie) gid=40 groups=40
    # root's uid=0
1.  id='id | gawk -F"[=()'  '{print $2}"      # get user id.
    echo your user id is: $id
2.  if (( id == 0 ))ㄱ      # [ $id. -eq 0 ] (See cd file: idcheck2)
     then
3.       echo "you are superuser."
4.  else
    echo "you are not superuser."
5.  fi
(The Comrnand Line)
6.  $ idcheck
    your user -id is:
    9496 you Are not superuser.
7.  $ su
     Password :
8.  # idcheck
     your user id is: 0
     you are superuser
```

설명

1. id지령의 출력이 gawk지령의 입력으로 되게 한다. gawk는 마당구분기호로 같기부호(=)와 열린괄호기호를 사용하며 출력으로부터 사용자 ID를 추출하여 변수 id에 출력을 대입한다.

2,3,4. id의 값이 0이면 3행을 실행한다. id가 0이 아니면 else명령문을 실행한다.

5. fi는 if지령의 끝을 현시한다.
6. idcheck스크립트는 UID가 9496인 현재사용자에 의해 실행된다.
7. su지령은 사용자를 root에로 절환한다.
8. #재촉문은 뿌리가 새로운 사용자라는것을 나타낸다. 뿌리의 UID는 0이다.

ㄱ. Bash 판본 2.x 에서는 실행되지 않는다.

5. if/elif/else명령문

if/elif/else지령은 여러개의 채택처리를 하게 한다. if다음의 지령이 실패하면 elif다음의 지령들을 검사한다. 지령이 성공이면 then명령문다음에 있는 지령들이 실행된다. elif지령이 실패이면 다음 elif지령이 검사된다. 성공된 지령이 없으면 else지령들을 실행한다. else블로크를 기정이라고 한다.

형식

```
if 지령
```

```
then
        지령(들)
elif지령
then
        지령(들)
elif 지령
then
        지령
else
        지령(들)
fi
```

실례 12-25

(The Script)

```
      #!/bin/bash
      # Scriptnanae: tellrne
      # Using the old-style test conmanct
1.  echo -n "How old are vou? "
      read age
2.  if [ Sage -lt 0 -o $age -gt 120 ]
      then
         echo   "Welcome to our planet! "
         exit 1
      fi
3.  if   [ $age -ge 0 -a $age -le 12 ]
      then
         echo "A child is a garden of verses"
      elif [ Sage -ge 12 -a Sage -le 19 ]
         then
echo "Rebel without a cause"
elif [ $age -ge 20 -a   Sage -le 29 ]
then
       echo "You got the world by the tail!!
elif [ Sage -ge   30 -a   Sage -le 39 ]
then
       echo "Thirty something..."
4.  else
      echo "Sorry I asked"
5.  fi
```

(The Output)

```
$ tellme
How old are you? 200
Welcome to our planet!

$ tellme
```

How old are you? ***13***
Rebel without a cause
$ tellme
How old are you? 55
Sorry I asked

설명

1. 사용자에게 입력을 요구한다. 입력을 변수 age에 대입한다.
2. 수값검사가 test지령에 의해 실행된다. age가 0보다 작거나 120보다 크면 echo지령이 실행되고 프로그람은 탈퇴상태 1을 가지고 끝난다. 대화형쉘재촉문이 나타난다.
3. 수값검사가 test지령에 의해 수행된다. age가 0과 갈거나 크고 12보다 작거나 갈으면 test지령은 탈퇴상태 0 즉 참으로 귀환되여 then다음의 명령문이 실행된다. 그외에는 프로그람조종을 elif로 보낸다. 검사가 거짓이면 다음 elif가 검사된다.
4. else구조는 기정이다. 우의 명령문들이 모두 참이 아니면 else지령들을 실행한다.
5. fi는 초기 if명령문을 끝낸다.

6. 파일시험

사용자가 스크립트들을 쓸 때 스크립트는 효력 있는 정확한 파일들을 요구하게 된다.

즉 파일들이 지정된 허가, 정확한 형태, 그외의 속성들을 가져야 한다는것이다. 해당한 스크립트작성에 필요한 부분을 검사하여 파일을 찾을수 있다.

표12-5. 파일시험연산자

검사연산자	참일때의 의미
-b파일이름	블로크특수파일
-c파일이름	문자특수파일
-d파일이름	등록부존재
-e파일이름	파일존재
-f파일이름	정규파일존재 하지만 등록부는 아니다.
-G파일이름	파일이 존재하면서 효력 있는 그룹을 가지고 있으면 참이다.
-g파일이름	그룹 ID를 설정한다.
-k파일이름	련결 비트를 설정한다.
-L파일이름	파일이 기호련결된다.
-P파일이름	파일은 이름 붙인 파이프이다.

(표계속)

검사연산자	참일때의 의미
-O파일이름	파일이 존재하며 효력 있는 사용자 ID를 가지고 있다.
-r파일이름	파일은 읽기가능하다.
-S파일이름	파일은 소케트이다.
-s파일이름	파일의 크기가 령이 아니다.
-tfd	fd(파일서술자)가 말단에서 열려 져 있으면 참이다.
-u파일이름	사용자 ID 비트를 설정한다.
-w파일이름	파일은 쓰기가능하다.
—x파일이름	파일은 실행가능하다.

실례 12-26

```
(The Script)
  #!/bin/bash
  # Using the old style test command
  # filename: perm_check
  file"./testing
1.  if [ -d $file ]
     then
     echo "$file is a directory"
2.  elif [ -f $file ]
     then
3.  if [ -r $file -a -w $file -a -x $file ]
     then        # nested if command
     echo "You have read,write,and execute \
     permission on $file."
4.  fi
5.  else
       echo "$file is neither a file nor a directory.”
6.  Fi
  ———————————————————————————————————

  #!/bin/bash
  # Using the new compound operator for test (( ))┐
  # filename: perm_check2
   file"./testing

7.  if [[ -d Sfile ]]
    then
```

```
echo "$file is a directory"
8.  elif [[ -f $file ]]
     then
9.         if [[ -r $file && -w $file && -x $file ]]
     then       # nested if command
echo "You have read,write,and execute \
permission on $file."
10.      fi
11.   else
        echo "$£ile is neither a file nor a directory.
fi
```

설명

1. 파일 testing이 등록부이면 testing is a directory를 인쇄한다.
2. 파일testing이 등록부가 아니면서 elss if 파일이 아니면 then을 실행한다.
3. 파일 testing이 읽기가능, 쓰기가능, 실행가능하면 then을 실행한다.
4. fi는 제일 마지막 if지령을 끝낸다.
5. else지령은 1행과 2행이 거짓이면 실행된다.
6. fi는 첫번째 if에로 간다.

ㄱ. 합성연산자를 쓴 새로운 형의 검사지령은 bash 판본 2.x우에서 실행되지 않는다.

7. null지령

두점으로 표시되는 null지령은 탈퇴상태 0을 되돌려 주고는 아무런 동작도 하지 않는 내부지령이다. 그것은 아무런 동작도 하지 않지만 필요하다.

프로그람이 then명령문다음에 무엇인가 요구되기때문에 오유통보문을 현시하려고 할 때 if지령다음에 써넣어 사용한다. null지령은 무한순환을 만들기 위한 loop지령에 인수로서 사용된다.

실례 12-27

```
(The Script)
#i/bin/bash
# filename: name_grep
1.  name=Tom
2.  if grep "$naine" databasefile >& /dev/null
       then
3.             :
4.      else
          echo "$1 not found in databasefile"
          exit 1
    fi
```

설명

1. 변수 name에 문자렬 Tom을 대입한다.
2. if지령은 grep지령의 탈퇴상태를 검사한다. 만약 Tom을 자료기지안에서 찾으면 null지령 즉 두점(:)이 실행되여 아무런 동작도 하지 않는다. 출력과 오유를 모두 /dev/null로 방향바꾸기 한다.
3. 두점은 null지령이다. 그것은 탈퇴상태 0으로 되돌아 가는 외에 아무것도 하지 않는다.
4. 만약 Tom을 찾지 못하면 실지로 수행하려고 하는것은 오유통보문을 인쇄하고 탈퇴하는것이다. else다음의 지령은 grep지령이 실패하면 실행된다.

실례 12-28

(The Command Line)
1. $ **DATAFILE=**
2. $ **: ${DATAFILE8=$HOME/db/data£ile}**
   $ **echo $DATAFILE**
   /home/jody/el1ie/db/datafi1e
3. $ **: ${DATAFILE:=$HOME/junk}**
   $ **echo $DATAFII.E**
   /home/jody/el1ie/db/datafi1e

설명

1. 변수 DATAFILE에 빈값을 대입한다.
2. 두점지령은 아무런 동작도 하지 않는 지령이다. 변경지시자(:=)는 변수에 대입할수 있거나 검사에서 사용되는 값을 돌려 준다. 이 실례에서 표현식은 아무런 동작도 하지 않는 지령에 인수로서 넘어 간다. 쉘은 변수바꿔넣기를 하는데 DATAFILE이 값을 가지고 있지 않으면 DATAFILE파일에 경로이름을 대입한다. 변수 DATAFILE이 영구적으로 설정되였다.
3. 변수가 이미 설정되였기때문에 변경지시자의 오른쪽에 있는 기정값으로 재설정되지 않는다.

실례 12-29

(The Script)
#i/bin/bash
1. *# Scriptname: wholenum*
   # Purpose:The expr command tests that the user eaters an
   # integer
   #
   echo "Enter an integer."
   read number
2. if **expr "$nuinber" + 0 >& /dev/null**
   then

```
3.          :
      else
4.          echo "You did not enter an integer value."
                  exit 1
5.  fi
```

설명

1. 사용자에게 옹근수입력을 진행할데 대해 요구한다. 수값을 변수 number에 대입한다.
2. expr지령은 표현식을 평가한다. 만약 더하기가 수행되면 수는 모든 수이며 expr는 성공이라는 탈퇴상태값을 되돌린다. 모든 출력을 비트바키트 /dev/null에로 방향바꾸기 한다.
3. expr가 성공하면 탈퇴상태 0을 되돌린다. 또한 두점지령은 아무러한 동작도 하지 않는다.
4. expr가 실패하면 령이 아닌 탈퇴상태값을 되돌린다. echo지령은 통보문을 현시하며 프로그람은 탈퇴한다.
5. fi는 if블로크를 끝낸다.

8. case지령

case지령은 if/elif지령들의 또다른 사용법으로 사용되는 다중분기지령이다. case변수의 값을 value1, value2와 같은 순서로 계속하여 case변수의 값과 일치되는 값을 찾을 때까지 대조해 나간다. case변수의 값과 일치하는 값이 있을 때 그 값다음에 있는 지령들을 2중반두점이 나타날 때까지 실행시킨다. 다음의 실행은 단어 esac(case를 거꾸로 쓴 단어)다음에서 부터 시작된다.

case변수와 일치되는 값이 없으면 프로그람은 *)의 다음에 있는 지령인 기정값으로서 ;; 또는 esac가 나타날 때까지 실행된다. *)는 if/else조건명령문에서 else와 같은 기능을 가지고 있다. case값들은 쉘의 통용기호와 2개의 값을 위해 수직기호(파이프기호)를 사용할수 있다.

형식

```
case 변수
변수1)
        지령(들)
             ;;
변수2)
        지령(들)
             ;;
*)
지령(들)
        ;;
esac
```

실례 12-30

```
(The Script)
  #!/bin/bash
  # Scriptname: xcolors
1.  echo –n "Choose a foreground color for your xterm window: "
     read color
2.  case "$color" in
3.  [Bb]1??)
4.      xterm –fg blue –fn terminal &
5.      ;;
6.  [Gg]ree*)
           xterm –fg darkgreen   -fn terminal &
           ;;
7.  red | orang)                  # The   vretical bar means "or"
           xterm –fg "$color"    -fn terminal &
           ;;
8.  *)
           xterm –fn terminal
           ;;
9.  esac
10.   echo "Out of case command"
```

설명

1. 사용자에게 입력을 요구하여 그 입력을 변수 color에 대입한다.
2. case지령은 표현식 $color를 평가한다.
3. color가 B 아니면 b로 시작되여 다음에 한개 문자 l, 그다음에 임의의 2개 문자로 되여있으면 case표현식은 첫번째 값과 대조한다. 값은 한개 닫긴 괄호로 끝난다. 통용기호는 파일이름확장에 사용되는 쉘메타문자들이다. xterm지령은 푸른 색으로 전경색을 설정한다.
4. 3행에 있는 값이 case표현식과 일치되면 명령문이 실행된다.
5. 2개의 반두점은 이 지령블로크에 있는 마지막지령다음에 요구된다. 반두점이 나타날 때 조종은 10행으로 넘어 간다. 스크립트는 반두점이 독자적인 행이므으로 스크립트를 오유수정하기가 쉽다.
6. case표현식이 G 혹은 g다음에 련이어 문자렬 ree이면서 임의의 개수의 다른 문자들로 끝나게 되면 xterm지령이 실행된다. 2중반두점은 명령문블로크를 끝내며 조종을 10행으로 넘긴다.
7. 수직막대기(|)는 or조건연산자로 사용된다. case표현식이 red 혹은 orange 둘중의 하나와 일치되면 xterm지령이 실행된다.
8. 이것은 기정값이다. 우에서 제시하는 값들가운데서 case표현식과 일치하는 값이 없으면 *)다음의 지령들이 실행된다.
9. esac명령문은 case지령을 끝낸다.
10. case값들가운데서 하나가 일치된 다음에 실행이 이 행에서부터 계속된다.

here문서와 case지령으로 차림표 만들기 here문서와 case지령은 일반적으로 함께 사용된다. here문서는 현시장치에 현시하려는 선택차림표를 만들기 위해 사용된다. 사용자에게 차림표항목들중 어느 하나를 선택하도록 하며 case지령은 해당한 지령을 실행하기 위해 선택된것을 다시 검사한다(실례 12-31).

실례 12-31

(From the *.bash_profile* File)

```
echo "Select a terminal type:
1.  cat <<- EMDIT
        1). unix
        2). xterm
        3). sun
2.  EHDIT
3.  read choice
4.  case "$choice" in
5.  1) TERM=unix
          export TERM
          ;;
2) TERM=xterm
          export TERM
          ;;
6.  3)   TERM=sun
        export TERM
      ;;
7.  esac
8.  echo "TERM is $TERM. "
```

(The Output)

```
$ . .bash_profile
Select a terminal type:
  1)  unix
  2)  xterm
  3)  sun
2    <-- User input
TERM is xterm.
```

설명

1. 스크립트의 토막이 등록가입할 때 .bash-profile에 넣어 지면 적당한 말단형태를 선택하게 한다. here문서는 선택차림표를 현시하기 위해 사용된다.
2. 사용자정의 ENDIT완료자는 here문서의 끝을 나타낸다.
3. read지령은 사용자입력을 변수 TERM에 기억시킨다.
4. case지령은 변수 TERM을 평가하여 닫기괄호앞에 있는 값들가운데서 어느 한개 값 즉 1, 2 혹은 *를 비교한다.
5. 첫번째로 검사되는 값은 1이다. 일치하면 말단은 unix로 설정된다. TERM 변수가 출력되여 자식쉘이 그것을 계승하도록 한다.

6. 기정값은 요구하지 않는다. TERM변수는 보통 등록가입할 때 /etc/profile 에 대입된다.
7. esac는 case지령을 끝낸다.
8. case지령을 끝내면 이 행이 실행된다.

제 6 절. 순환지령

순환지령들은 한개 지령이나 여러개의 지령들을 지정된 회수만큼 혹은 일정한 조건을 만족시킬 때까지 실행시키는데 쓴다. bash쉘에는 3가지 순환형태 즉 for순환, while 순환, until순환이 있다.

1. for지령

for순환지령은 지령을 항목에 있는 회수만큼 실행시키는데 사용한다. 실례로 파일들이나 사용자이름목록에 대하여 같은 지령들을 실행시키기 위해 순환지령을 사용할수 있다. for지령다음에는 사용자정의변수가 놓이고 그다음에 예약어 in이, 그다음에 단어들의 목록이 놓인다.

첫번째 순환에서는 단어목록에 있는 첫번째 단어가 변수에 대입되고 그다음에 목록에서 밀려 난다. 단어를 변수에 대입하면 순환본체에 들어 가 do와 done예약어사이에 있는 지령들을 실행한다. 다음번 순환에서는 두번째 단어를 변수에 대입하며 우와 같은 과정들이 계속 반복된다. 순환본체는 do예약어로 시작하여 done예약어로 끝난다. 목록에 있는 모든 단어들이 밀려 나면 순환은 끝나며 프로그람조종권은 done예약어다음에로 넘어 간다.

형식

```
for variable in word_list
do
        command(s)
done
```

실례 12-32

```
(The Script)
   #!/bin/bash
   #Scriptname:forloop
1. for pal in Tom Dick Harry Joe
2. do
3. echo "Hi $pal"
4. done
5. echo "Out of loop"

(The Output)
$ forloop
Hi Tom
Hi Dick
```

Hi Harry
Hi Joe
Out of loop

설명

1. for순환은 이름목록 Tom, Dick, Harry, Joe를 하나씩 리용한 다음 밀기하면서 반복한다(왼쪽에서부터 밀기하면서 그 값을 사용자정의변수 pal에 대입한다.). 모든 단어들이 밀기되여 단어목록이 비게 되면 순환은 끝나고 실행은 done예약어다음부터 시작된다. 첫번째 순환에서는 변수 pal에 단어 Tom을 대입하고 두번째 순환에서는 pal에 단어 Dick가, 그다음 순환에서는 Harry가, 그다음번 순환에서는 Joe를 대입한다.
2. do예약어는 단어목록다음에 놓이게 된다. 만약 같은 행에 쓰려면 반드시 반두점으로 구분해야 한다. 즉
   for pal in Tom Dick Harry Joe; do
3. 이 부분은 순환본체이다. Tom을 변수값으로 대입한 다음 순환본체에 있는 지령(즉 do와 done예약어사이의 모든 지령)들이 실행된다.
4. done예약어로 순환을 끝낸다. 단어목록의 마지막단어(Joe)가 대입된 다음 밀기되면 순환에서 벗어 난다.
5. 순환에서 벗어 나면 조종권이 넘어 온다.

실례 12-33

```
(The Command Line)
1.  $ cat mylist
     tom
     patty
     ann
     jake

(The Script)
   #!/bin/bash
   #Scriptname:mailer
2.  for person In $(cat mylist)
     do
3.  mail $person < letter
     echo $person was sent a letter.
4.  done
5.  echo "The letter has been sent."
```

설명

1. mylist라고 하는 파일의 내용을 현시한다.
2. 지령바꿔넣기가 진행되고 mylist의 내용이 단어목록으로 된다. 첫 순환에서는 tom을 변수 person에 대입한 다음 한자리 밀기되여 patty로 교체되며 이와 같은 과정을 계속 반복한다.
3. 순환본체에서는 매 사용자에게 letter파일을 우편으로 보낸다.

4. done예약어는 순환반복의 끝을 나타낸다.
5. 목록에 있는 모든 사용자들에게 우편을 보내고 순환이 끝나면 이 행을 실행한다.

실례 12-34

(The Script)

```
#!bin/bash
#Scriptname:backup
#Purpose:Create backup files and store
#them In a backup directory.
#
1.  dir=/home/jody/ellie/backupscript
2.  for file In memo[1-5]
     do
3.       if [ -f $file]
     then
           cp $file $dir/$file.bak
         echo "$file is backed up in $dir"
   fi
4.  done
```

(The Output)

```
memo1 Is backed up In /home/jody/ellie/backupscripts
memo2 Is backed up In /home/jody/ellie/backupscripts
memo3 Is backed up In /home/jody/ellie/backupscripts
memo4 Is backed up In /home/jody/ellie/backupscripts
memo5 Is backed up In /home/jody/ellie/backupscripts
```

설명

1. 변수 dir에 여벌스크립트(backup script)가 기억되여 있는 등록부가 대입된다.
2. 단어목록은 memo로 시작하고 1~5까지의 수들로 끝나는 이름을 가진 현재 작업등록부의 모든 파일로 이루어 진다. 순환의 매 반복에서 변수 file에 모든 파일이름들을 한번에 하나씩 대입한다.
3. 순환본체에 들어 갈 때 파일의 존재성여부를 검사한다. 있으면 그 파일의 본래이름에 확장자 .bak를 붙여서 등록부 /home/jody/ellie/backup scripts에 복사한다.
4. done은 순환의 끝을 나타낸다.

단어목록에 있는 $*와 $@변수 $*와 $@가 확장될 때 인용부호안에 있지 않는한 그 기능들은 같다. $*는 한개 물음표, $@는 분리된 단어들의 목록과 같다.

실례 12-35

(The Script)

```
#!/bin/bash
```

```
#Scriptname:greet
1.  for name in $*              # same as for name In $@
2.  do
        echo Hi $name
3.  done
(The Comand Line)
   $ greet Dee Bert Lizzy Tommy
   Hi Dee
   Hi Lizzy
   Hi Tommy
```

설명

1. $*와 $@는 모든 위치파라메터들의 목록으로서 이 경우에는 지령행에서 넘어 온 인수들인 Dee, Bert, Lizzy와 Tommy들로 확장된다. 목록에 있는 매 이름을 차례로 for순환에 있는 변수 name에 대입한다.
2. 순환본체에 있는 지령들을 목록이 빌 때까지 실행한다.
3. done예약어는 순환본체의 끝을 나타낸다.

실례 12-36

```
(The Script)
#!/bin/bash
# Scriptname:permx
1.  for file                # Empty wordlist
    do
2.  if [ -f $file –a ! –x $file ]or if [[ -f $file && ! –x $file ]]a
    then
3.              chmod +x $file
       echo $file now has execute permission
    fi
    done

(The command Line)
4.  $ permx *
    adden now has execute permission
    checken now has execute permission
    doit now has execute permission
```

설명

1. 만일 for순환에 단어목록이 주어 지지 않으면 위치파라메터들을 반복한다. 이것은 $*에서 for file과 같다.
2. 파일이름을 지령행에서 넘겨 받는다. 쉘은 *를 현재작업등록부안의 모든 파일이름들로 한다. 파일이 보통파일이고 실행허가가 없으면 3행을 실행한다.
3. 실행허가를 처리하려는 매 파일에 추가한다.

4. 지령행에서 쉘은 *를 통용기호로 해석하여 *대신에 현재등록부에 있는 모든 파일들을 교체한다. 파일들을 Permx스크립트에 인수들로서 넘긴다.

2. while지령

while지령은 뒤에 오는 식의 값을 평가하고 탈퇴상태가 0이면 순환본체에 있는 지령들을 실행한다. done예약어가 나타나면 조종은 순환의 꼭대기로 되돌아 가고 while지령은 지령의 탈퇴상태를 다시 검사한다. while지령에 의해 평가되는 지령의 탈퇴상태가 0이 아닐 때까지 순환은 계속된다. 탈퇴상태가 0이 아니면 프로그람은 done예약어다음부터 실행을 시작한다.

형식

```
While 지령
do
     지령(들)
done
```

실례 12-37

(The Script)

```
#!/bin/bash
#Scriptname:num
1.  num=0         # Initialize num
2.  while (( $num < 10 ))a        #or while [ num –lt 10 ]
     do
     echo –n "$num "
3.       let num+=1                    # Interement num
     done
4.  echo –e "\ nAfter loop exits, continue running here"
```

(The Output)

```
0 1 2 3 4 5 6 7 8 9
4. After loop exits,continue running here
```

설명

1. 초기화단계로서 변수 num에 0을 대입한다.
2. while지령다음에 let지령이 놓인다. let지령은 산수연산을 평가하고 조건이 만족되면 탈퇴상태가 0으로 된다. 즉 num이 10보다 작으면 순환에 들어 간다.
3. 순환본체에서 num값은 하나씩 증가된다. 만약 num값이 변하지 않는다면 순환은 무한히 반복되거나 프로세스가 끝날 때까지 계속된다.
4. 순환에서 벗어 나면 echo지령(-e선택을 가진)이 행바꾸기문자와 문자렬을 인쇄한다.

실례 12-38

(The Script)

```
 #!/bin/bash
# Scriptname: quiz
1.   echo "Who was the 2nd U.S. president to be impeached?"
      read answer
2.   while [ "$answer" != "Bill Clinton" ]
3.   do
         echo   "Wrong try again!"
4.          read answer
5.   done
6.   echo You got it!
```

(The Output)

```
$ quiz
Who was the 2nd U.S.persident to be impreached? Ronald Reagan
Wrong try again!
Who was the 2nd U.S.persident to be impreached? I give up
Wrong try again!
Who was the 2nd U.S.persident to be impreached? Bill Clinton
```

설명

1. echo지령은 사용자에게 Who was the 2nd U.S. president to be impeached?라는 재촉문을 현시한다. read지령은 사용자의 입력을 기다린다. 입력된 값은 변수 answer에 기억된다.
2. while순환이 시작되여 중괄호로 표시한 test지령이 식을 검사한다. 변수 answer가 Bill Clinton과 같지 않으면 순환을 시작하여 do와 done사이에 있는 지령들을 실행한다.
3. do예약어는 순환본체의 시작이다.
4. 사용자에게 재입력을 요구한다.
5. done예약어는 순환본체의 끝을 의미한다. 조종을 다시 while순환의 꼭대기로 되돌려 보내고 식을 다시 검사한다. Answer가 Bill Clinton이 아니면 순환을 계속 반복한다. 사용자가 Bill Clinton을 입력하면 순환이 끝나게 된다. 프로그람의 조종은 6행으로 넘어 간다.
6. 순환이 끝나면 여기에서 조종이 시작된다.

실례 12-39

(The Script)

```
$ cat sayit
     #!/bin/bash
     #Scriptname: sayit
     echo   Type q to quit.
     go=start
```

```
1.  while [ -n "$go" ]      # Make sure to double quote the variable
    do
2.      echo -n I love you.
3.      read word
4.      if  [[ "$word" = Qq]]
        then          # [ "$word" =q-o  "$word" =Q] Old style
            echo "I'll always love you!"
               go=
            fi
    done
```

(The Output)
$ **sayit**
Type q to quit.
I love you. ← *When user presses Enter,the program continues*
I love you.
I love you.
I love you.
I love you.q
I'll always love you!
$

설명

1. while다음에 있는 지령을 실행하고 그 탈퇴상태를 검사한다. test지령에 대한 n 추가선택은 문자렬이 빈값이 아닌인가를 검사한다. 변수 go는 초기값을 가지므로 검사가 성공되여 출력상태를 0으로 한다. 만일 변수 go가 2중인용부호안에 있지 않고 또한 그 변수가 빈값이라면 test지령은 다음과 같은 통보문을 내보낸다.
   go: test: argument expected
2. 순환을 시작한다. 문자렬 I love you가 현시장치에 현시된다.
3. read지령은 사용자입력을 요구한다.
4. 식을 평가한다. 사용자가 q나 Q를 입력한 경우 문자렬 I'll always love you!가 현시되고 변수 go를 빈값으로 설정한다. while순환을 다시 시작할 때 변수가 빈값이므로 시험은 실패하며 순환은 끝난다. 조종은 done예약어 다음에로 이행한다. 이 실례에서 스크립트는 더 실행할 행이 없으므로 중단된다.

3. until지령

until지령은 while지령과 류사한데 다만 until다음의 지령이 실패(거짓)일 때만 즉 명령이 0아닌 탈퇴상태를 돌려 줄 때만 순환이 진행된다. done예약어가 나타나면 조종은 순환의 꼭대기로 이행하며 until지령은 그 지령의 탈퇴상태를 다시 검사한다. until지령에 의해 평가되는 지령의 탈퇴상태가 0으로 될 때까지 순환은 계속된다. 탈퇴상태가 0으로 되면 순환은 탈퇴되며 프로그람실행은 done예약어다음부터 시작된다.

형식

```
until command
do
        command(s)
done
```

실례 12-40

```
#!/bin/bash
1.  until who | grep linda
2.  do
     sleep 5
3.  done
     talk linda@dragonwings
```

설명

1. until순환은 파이프에 있는 마지막지령인 grep의 탈퇴상태를 검사한다. who지령은 이 콤퓨터에 누가 가입하였는가를 표시하며 그 출력을 grep로 파이프한다. grep지령은 사용자 linda를 찾았을 때에만 0탈퇴상태를 되돌려 준다.
2. 만약 사용자 linda가 가입하지 않았다면 순환본체에 들어 가 프로그람은 5s동안 정지상태에 놓인다.
3. linda가 가입하면 grep명령의 탈퇴상태는 0으로 되고 done예약어다음의 명령이 실행된다.

실례 12-41

```
(The Script)
$ cat hour
#!/bin/sh
# Scriptname: hour
            1.          let hour=0
2. until (( hour > 24 ))a       # or [ $hour -gt 24 ]
do
3.        case "$hour" in
    [0-9] | 1[0-1])     echo    "Good morning!"
        ;;
    12)         echo    "Lunch time."
        ;;
    1[3-7]) echo    "Siesta time."
        ;;
    *)          echo    "Good night."
        ;;
    esac
4.        let hour+=1       # Don't forget to increment the hour
5. done
```

(The Output)
$ **hour**
Good moring!
Good moring!

Lunch Time.
Siesta time.

Good night.

설명

1. 변수 hour를 0으로 초기화한다.
2. let명령은 시간이 24보다 크거나 같은가를 비교한다. 만약 시간이 24보다 작으면 순환이 시작된다. 조건이 참일 때까지 순환은 계속 반복된다.
3. case명령은 변수 hour의 값을 검사하여 매 case명령문과 일치되는가를 검사한다.
4. hour변수는 let지령에 의해 증가된다. 조종이 순환의 꼭대기로 되돌아 가기 전에 let지령에 의해 증가된다.
5. done지령은 순환본체의 끝을 나타낸다.

4. select지령과 차림표

here document는 차림표를 만드는 손쉬운 방법이지만 bash는 select순환이라고 하는 다른 순환도구를 받아 들여 차림표를 만드는데 기본적으로 사용하고 있다. 수자들로 목록화된 항목들의 차림표를 표준오유로 현시한다. PS3재촉문은 사용자에게 입력을 재촉하는데 쓰이는데 기정값으로 PS3은 #?이다. PS3재촉문이 현시된 다음 쉘은 사용자입력을 기다린다. 입력값은 차림표목록에 있는 수들중의 어느 하나이다. 입력은 쉘의 특수변수인 REPLY에 기억된다. REPLY변수에 있는 수자는 선택목록에 있는 괄호의 오른쪽에 있는 문자렬과 련관된다.

case지령은 select지령과 같이 리용되여 사용자가 차림표에서 한개를 선택하고 그에 기초하여 지령을 실행할수 있도록 한다. LINES와 COLUMNS변수들은 말단에 현시된 차림표항목들의 배치상태를 결정하는데 쓴다(이 변수들은 bash2.x는 있지만 그이전 판본에는 없는것이다. 만일 그것들이 정의되지 않았다면 .bash_profile에서 그것들을 정의하고 넘길수 있다.). 출력을 표준오유로 현시하는데 그 매 항목의 앞에는 수자와 닫긴괄호가 있고 PS3재촉문을 차림표의 맨 밑에 현시한다. select지령이 순환지령이기때문에 순환에서 벗어 나기 위해서는 break지령을 쓰며 스크립트에서 탈퇴하기 위해서는 exit지령을 써야 한다.

형식

```
select var in wordlist
do
        지령(들)
done
```

실례 12-42

```
(The Script)
   #!/bin/bash
   # Program name: runit
1.  PS3="Select a program to execute:
2.  select program   in 'ls -F' pwd date
3.  do
4.  $program
5.  done

(The Command Line)
Select a program to execute:2
   1) ls  □F
   2) pwd
   3) date
/home/ellie
select a program to execute:1
   1) ls  □F
   2) pwd
   3) date
12abcrty abc12 doit* progs/ xyz
Select a program to execute:3
   1) ls  □F
   2) pwd
   3) date
Sun Mar 12 13:28:25 PST 2001
```

설명

1. PS3재촉문에 select순환이 표시하는 차림표아래에 있는 문자렬이 대입된다. 이 재촉문은 기정으로 $#이며 표준오유인 현시장치에 보낸다.
2. select순환은 program이라는 변수와 차림표에 표시되는 세단어목록인 ls -F, pwd, date로 이루어 진다. 이 목록에 있는 단어들은 UNIX지령들인데 임의의 다른 단어들 실례로 red, green, yellow, cheese, bread, milk, crackers 들이여도 된다. 만약 단어에 공백이 있으면 괄호로 묶어 주어야 한다. 실례로 'ls -F'와 같이 한다.
3. do예약어는 순환본체의 시작을 의미한다.
4. 사용자가 차림표에 있는 번호를 선택할 때 그 번호는 괄호가 붙은 번호다음에 있는 단어와 같다. 실례로 사용자가 번호 2를 선택하면 그것은 pwd로 되며 pwd는 변수 program에 기억된다. $program은 실행지령 pwd와 같다. 지령이 실행된다.
5. done예약어는 select순환에 있는 명령문들의 끝을 나타낸다. 조종은 순환의 꼭대기로 돌아 간다. 이 순환은 사용자가 ˆC를 누를 때까지 계속 실행된다.

실례 12-43

```
(The Script)
   #!/bin/bash
   # Program name: goodboys
1.  PS3="Please choose one of the three boys : "
2.  Select choice in tom dan guy
3.  do
4.        case $choice in
           tom)
           echo Tom is a cool dude!
5.            break;;                    #break out of the select loop
6.       dan | guy )
           echo Dan and Guy are both sweethearts.
           break;;
           *)
7.            echo " $REPLY is not one of your choices " 1>&2
              echo "Try again. \ n"
              ;;
8.      esac
9. done

(The Command Line)
$ goodboys
            1). tom
            2). dan
            3).guy
        Please choose one of the three boys : 2
        Dan and Guy are both wonderful.

        $ goodboys
           1)        tom
           2)        dan
           3)        guy
        Please choose one of the three boys : 4
        2 is not one of your choice
        Try again.
        Please choose one of the three boys : 1
        Tom is a ool dude!
        $
```

설명

1. 두번째 행에 있는 select순환으로 만들어 진 차림표우에 PS3재촉문을 현시한다.
2. select순환에 들어 간다. 목록에 있는 단어들을 번호가 붙은 차림표로 현시되게 한다.
3. 순환본체는 이 행에서부터 시작된다.

4. 목록에 있는 첫번째 값을 변수에 대입한 다음 목록에서 밀기를 진행하여 다음항목이 첫번째 값으로 된다.
5. break명령문은 순환조종을 아홉번째 행다음에로 옮긴다.
6. dan 혹은 guy가 선택되면 다음에 있는 echo지령을 실행하며 그다음에 break지령을 실행하여 조종을 9행으로 보낸다.
7. 내부 REPLY변수는 현재목록항목 즉 1, 2, 3과 같은 번호를 가지고 있다.
8. case지령의 끝을 나타낸다.
9. done은 select순환의 끝을 나타낸다.

실례 12-44

(The Script)

```
     #!/bin/bash
     # Program name: ttype
     # Purpose: set the terminal type
     # Author: Andy Admin
1.   COLUMNS=60
2.   LINES=1
3.   PS3="Please enter the terminal type: "
4.   Select choice in wyse50 vt200 xterm sun
     do
5.     case $REPLY in
        1)
6.
        export TERM=$choice
        printf "TERM=$choice \ n"
        break;;             # break out of the select loop
        2 | 3 )
        export TERM=$choice
        printf "TERM=$choice \ n"
        break;;
        4)
        export TERM=$choice
 printf "TERM=$choice \ n"
 break;;
        *)
7.              echo □e "$REPLY is not a valid choice. Try again \ n" 1>&2
8.              REALY=   # Causes the menu to be redisplayed
                ;;
     esac
9.   done
```

(The Command Line)

$ ttype

1) wyse50 2) vt200 3) xtexm 4) ***sun***

Please enter the terminal type : 4
TERM=sun

$ ttype

1) wyse50 ***2) vt200*** ***3) xterm*** ***4) sun***
Please enter the terminal type : 3 TERM=xterm

$ ttype **1) wyse50 2) vt200 3) xterm 4) sun**
Please enter the terminal type : 7
7 is not a valid choice. Try again.

1) wyse50 2) vt200 3) xterm 4) sun
Please enter the terminal type: 2

TERM=vt200

설명

1. COLUMNS변수는 select순환으로 만들어 진 차림표에서 렬형태로 현시되는 말단현시너비를 설정한다. 기정값은 80이다.
2. LINES변수는 말단에서 select차림표의 수직현시를 조종한다. 기정값은 24행이다. LINES값을 1로 변화시킬 때 차림표항목을 마지막실례에서와 같이 수직으로 표시하는것이 아니라 한 행으로 표시한다.
3. PS3재촉문이 설정되여 차림표선택아래에 나타난다.
4. select순환은 4개의 선택요소 즉 wyse50, vt200, xterm, sun으로 된 차림표를 현시한다. REPLY변수에 들어 있는 사용자의 응답에 기초한 값들중의 하나를 변수 choice에 대입한다. 만일 REPLY가 1이면 choice에 wyse50을 대입하고 REPLY가 2이면 choice에 vt200을 대입하며 REPLY가 3이면 choice에 xterm을 대입하고 REPLY가 4이면 choice에 sun을 대입한다.
5. REPLY변수는 사용자의 입력선택과 같다.
6. 말단장치의 형태를 대입하고 반출하며 인쇄한다.
7. 사용자가 1~4까지를 제외한 다른 값을 입력했을 때는 다시 입력할것을 요구한다. 그러나 차림표는 나타나지 않고 PS3재촉문만 나타난다.
8. REPLY가 빈값으로 설정되면 즉 REPLY=이면 차림표를 다시 현시한다.
9. Select순환의 끝이다.

5. 순환조종

어떤 조건이 발생하면 순환에서 벗어 나 순환의 꼭대기로 돌아 가거나 무한순환을 중지할 필요가 있다. bash쉘에는 이와 같은 처리를 위한 순환조종지령들을 가지고 있다.

shift지령 shift지령은 지적된 회수만큼 파라메터목록을 왼쪽으로 밀기한다.

shift지령에 인수가 없으면 파라메터목록을 왼쪽으로 한번만 밀기한다. 목록을 한번 밀기하면 그 파라메터를 완전히 삭제한다. shift지령은 위치파라메터목록을 반복할 때 while순환에서 자주 쓰인다.

형식

shift [n]

실례 12-45

```
(Without a Loop)
(The Script)
#!/bin/bash
#Scriptname:shifter
1.  set joe mary tom sam
2.  shift
3.  echo $*
4.  set $(date)
5. echo $*
6.  shift 5
7.  echo $*
8.  shift 2
```

```
(The Output)
3.  mary tom sam
5.  The Mar 16 10:00:12 PST 2001
7.  2001
8.  shift:shift count must be=<$#
```

설명

1. set지령은 위치파라메터들을 설정한다. $1에 joe를, $2에 mary를, $3에 tom을, $4에는 sam을 대입한다. $*는 모든 파라메터들을 나타낸다.
2. shift지령은 파라메터를 왼쪽으로 밀기한다. 즉 joe가 밀기된다.
3. 파라메터목록을 밀기한 다음에 인쇄한다.
4. set지령은 UNIX의 date지령의 출력으로 위치파라메터를 재설정한다.
5. 새로운 파라메터목록이 인쇄된다.
6. 이때 목록을 왼쪽으로 5번 밀기한다.
7. 새로운 파라메터목록을 인쇄한다.
8. 주어 진 파라메터개수이상으로 밀기하면 쉘은 표준오유통보를 보내여 shift지령이 초과되는 파라메터들을 밀기할수 없다는것을 알려 준다. $#는 위치파라메터의 총 개수이다. bash 2.x판본에서는 아무런 오유통보도 생기지 않는다.

실례 12-46

```
(With a Loop)
(The Script)
   #!bin/bash
   #Name: doit
   #Purpose: shift through command line arguments
1.  while (( $# > 0 ))a   #   or   [ $# -gt 0]
```

```
       do
2.       echo $*
3.       shift
4.    done
```

(The Command Line)
$ **doit a b c d e**
a b c d e
b c d e
c d e
d e
e

설명

1. let지령은 수식을 검사한다. 위치파라메터들의 개수($#)가 0보다 크면 순환본체에 들어 간다. 위치파라메터들은 지령행에서 인수로서 넘어 온다. 인수는 5개이다.
2. 모든 위치파라메터들을 인쇄한다.
3. 파라메터목록을 왼쪽으로 한번 밀기한다.
4. 이 행은 순환본체의 끝이므로 조종을 순환부의 꼭대기로 돌려 보낸다. 매 순환에 들어 갈 때마다 shift지령은 파라메터목록을 하나씩 감소시킨다. 첫번째 밀기를 진행한 다음 $#(위치파라메터들의 수)는 4개로 된다. $#가 0으로 되면 순환은 끝나게 된다.

실례 12-47

(The Script)

```
   #!/bin/bash
   # Scriptname: dater
   # Purpose: set positional parameters with the set command
   # and shift through the parameters.
1.    set $(date)
2.    while (( $# > 0 ))     #   or   [ $# -gt 0 ] Old style
   do
3.       echo $1
4.       shift
      done
```

(The Output)
$ **dater**
Wed
Mar
15
19:25:00
PST
2001

설명

1. set지령은 date지령을 출력하여 그것을 위치파라메터 $1~$6에 대입한다.
2. while지령은 위치파라메터들의 수($#)가 0보다 큰가 작은가를 검사하고 크다면 순환본체에 들어 가게 한다.
3. echo지령은 첫번째 위치파라메터인 $1의 값을 현시한다.
4. Shift지령은 파라메터목록을 왼쪽으로 한번 밀기한다. 매 순환은 목록이 빌 때까지 밀기한다. 그때 $#는 0으로 되며 순환도 끝나게 된다.

break지령 내부 break지령은 순환에서 즉시 탈퇴할 때 쓴다. 그러나 프로그람은 완료되지 않는다(프로그람을 완료하려면 exit지령을 쓴다.). break지령이 실행된 다음 조종은 done예약어다음에로 옮겨 간다. break지령은 제일 안쪽 순환으로부터 탈퇴되므로 사용자가 겹쌓인 순환을 리용할 때 break지령은 수자인수로 특징 짓는 바깥순환에서 탈퇴하도록 한다. 가령 3개의 겹쌓인 순환이 있을 때 제일 바깥쪽 순환은 순환번호 3, 다음 안쪽은 순환번호 2, 제일 안쪽은 순환번호 1이다. break지령은 무한순환에서 탈퇴할 때 효과적이다.

형식

```
break [n]
```

실례 12-48

```
   #!/bin/bash
   #Scriptname:loopbreak
1. while true; do
2.     echo Are you ready to move on\ ?
        read answer
3.     if   [[ "$answer" == [Yy] ]]
          then
4.        break
5.     else
            ......commands...
          fi
6. done
7. print "Here we are"
```

설명

1. true지령은 항상 상태 0을가지고 탈퇴하는 UNIX지령이다. 이것은 무한순환으로 시작하는데 자주 쓰인다. do명령문을 while지령과 같은행에 놓을수 있는데 이 경우에는 반두점으로 구분해 준다. 순환본체에 들어 간다.
2. 사용자에게 입력을 요구한다. 사용자입력을 변수 answer에 대입한다.
3. 만일 $answer가 Y 또는 y이면 조종은 4행으로 이행한다.
4. break지령을 실행하면 순환에서 탈퇴하며 조종은 7행으로 이행하여 Here we are를 인쇄한다. 사용자로부터 Y 또는 y입력을 받을 때까지 프로그람은 입력을 계속 요구한다.

5. 3행에서 검사가 실패하면 else지령을 실행한다. 순환본체가 done예약어에서 끝나면 조종은 1행에 있는 while지령에서부터 다시 시작한다.
6. 순환본체의 끝이다.
7. break지령이 실행된 다음 조종은 이 행에서 시작한다.

continue지령 continue지령은 일정한 조건을 만족하면 순환의 꼭대기로 조종을 돌려 보낸다. continue지령다음에 있는 모든 지령들이 무시된다. 만일 겹쌓인 순환이라면 continue지령은 제일 안쪽 순환에 조종을 돌려 보낸다. 인수가 수로 주어 졌다면 조종은 임의의 순환의 꼭대기에서 시작할수 있다. 가령 3겹쌓인 순환이 있다면 제일 바깥순환은 순환번호 3, 다음 안쪽순환은 순환번호 2, 제일 안쪽 순환은 순환번호 1이다.[4]

형식

```
Continue[n]
```

실례 12-49

```
(The mailing List)
   $ cat mail_list
   ernie
   john
   richrad
   mealnie
   greg
   robin

(The Script)
   #!/bin/bash
   #Scriptname:mailem
   #purpose: To send a list
1.   for name in $(cat mail_list)
      do
2.     if [[$name==richard]]; then
3.           continue
               else
4.             mail $name < memo
            fi
5.   done
```

설명

1. 지령을 $(cat mail_list)또는 'cat mail_list'로 대입한 다음 for순환은 mail_list파일에 있는 목록의 이름들을 반복한다.
2. 이름이 richard로 되는 경우 continue지령이 실행되며 조종은 순환식을 평가하는 순환의 꼭대기로 되돌아 간다. richard는 이미 목록에서 밀려 났기 때문에 다음 사용자 melani가 변수 name에 대입된다. 낡은 형식은 다음과

4. continue 지령이 순환번호보다 큰 수로 주어 지면 순환은 탈퇴한다.

갈다.

if [“$name”=richard] ; then

3. continue지령은 순환본체에 있는 나머지 지령들을 뛰여 넘어 조종을 순환의 꼭대기로 돌려 보낸다.
4. richard를 제외하고 목록에 있는 모든 사용자들에게 memo파일을 우편으로 보낸다.
5. 순환본체의 끝이다.

겹쌓인 순환과 순환조종 겹쌓인 순환을 리용할 때 break와 continue지령은 옹근수값을 인수로 하여 조종이 안쪽 순환에서 바깥쪽 순환에로 갈수 있게 한다.

실례 12-50

```
(The Script)
#!/bin/bash
# Scriptname: months
1.  for month in Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
    do
2.  for week in   1 2 3 4
    do
            echo -n "Processing the month of $month. OK? "
            read ans
3.         if [ "$ans" = n   -o -z "$ans" ]
           then
4.              continue 2
            else
       echo -n "Process   week $week of $month? "
       read ans
    if [ "$ans" = n -o -z "$ans"   ]
       then
5.                      continu
            else
                  echo "Now processing week $week of $month."
                     sleep 1
                     # Commands go here
                  echo   "Done processing..."
            fi
       fi
6.         done
7.  done
```

```
(The Output)
$ months
Procossing the month of Jan. OK?
Procossing the month of Feb. OK? Y
Process week 1 of Feb? y
Now processing week 1 of Feb.
```

Done processing...
Procossing the month of Feb. OK? Y
Process week 2 of Feb? y
Now processing week 2 of Feb.n
Done processing...
Procossing the month of Feb. OK? N
Procossing the month of Mar. OK? N
Procossing the month of Apr. OK? N
Procossing the month of May. OK? N

설명

1. 바깥쪽 for순환이 시작된다. 이 순환에서 먼저 month에 Jan을 대입한다.
2. 안쪽 for순환을 시작한다. 이 순환에서는 먼저 week에 1을 대입한다. 안쪽 순환을 다 끝내야 바깥쪽 순환으로 갈수 있다.
3. 사용자가 n 또는 Enter건을 누르면 4행이 실행된다.
4. 인수 2를 가진 continue지령은 두번째 바깥쪽 순환의 꼭대기에서 조종을 시작한다. 인수가 없는 continue지령은 제일 안쪽 순환의 꼭대기로 조종을 돌려 보낸다.
5. 조종은 제일 안쪽 for순환에로 돌아 간다.
6. done은 제일 안쪽 순환을 끝낸다.
7. done은 제일 바깥쪽 순환을 끝낸다.

6. I/O방향바꾸기 및 자식쉘

입력은 파일로부터 순환을 파이프 혹은 방향바꾸기할수 있다. 출력도 마찬가지이다. 쉘은 자식쉘을 기동하여 I/O방향바꾸기와 파이프를 조종한다. 순환안에서 정의된 임의의 변수들은 순환이 끝나면 스크립트뒤부분에서 미정으로 된다.

순환출력을 파일로 방향바꾸기 bash순환에서의 출력을 현시장치로가 아니라 파일로 보낼수 있다. 이것을 실례 12-51에 보여 주었다.

실례 12-51

(The Command Line)
1. $ **cat memo**
   abc
   def
   ghi

(The Script)
```
   #!/bin/bash
   # Program name:numberit
   # Put line numbers all lines of memo
2. if let $(($# < 1 ))
   then
3. echo “Usage: $0 filename “ >&2
```

```
        exit 1
        fi
4.  count=1                          # Initialize count
5.  cat $1 | while read line
# Input is coming from file provided at command line
do
6.      let $((count == 1)) && echo "Processing file $1..." > /dev/tty
7.      echo –e "$count\ t$line"
8.      let count +=1
9.  done > tmp$$                     # Output is going to a temporary file
10.   mv tmp$$ $1
(The Command Line)
11.   $ numberit memo
Processing file memo...
12.   $ cat memo
      1 abc
      2 def
      3 ghi
```

설명

1. memo파일의 내용을 현시한다.
2. 사용자가 스크립트를 실행하고 있을 때 지령행에 인수를 주지 않으면 인수의 개수($#)는 1보다 작아 지며 오유통보가 나타난다.
3. 인수의 개수가 1보다 작으면 사용법통보문을 stderr(>&2)에로 보낸다.
4. count변수에 1이 대입된다.
5. UNIX의 cat지령은 $1에 보존된 파일이름의 내용을 현시하며 출력은 while 순환으로 파이프된다. read지령의 첫번째 순환에는 파일의 첫 행이, 다음번 순환에는 파일의 두번째 행이 대입되며 이와 같은 과정이 계속된다. read 지령은 읽기입력이 성공하면 0탈퇴상태를, 실패하면 1을 돌려 준다.
6. count의 값이 1이면 echo지령을 실행하여 출력을 현시장치 /etc/tty에로 보낸다.
7. echo지령은 count값을 인쇄한 다음 파일에 있는 행을 인쇄한다.
8. count를 1만큼 증가시킨다.
9. 전체 순환의 출력 즉 $1에 있는 파일의 첫번째 행을 제외한 매 행을 파일 tmp$$로 방향바꾸기한다. 첫번째 행은 말단 /dev/tty에로 방향바꾸기 한다.[5]
10. tmp파일은 $1로 대입된 파일이름으로 변경된다.
11. 프로그람이 실행된다. 처리하려는 파일이름은 memo이다.
12. 파일 memo는 스크립트가 끝난후 현시되여 행번호가 매 행의 앞에 붙었다는것을 보여 준다.

순환출력을 UNIX지령으로 파이프하기 출력을 다른 지령(들)으로 파이프하거나 파

5. $$는 현재 쉘의 PID수자로 확장된다. 파일이름에 이 수자를 추가하여 파일이름을 유일하게 한다.

일로 방향바꾸기할수 있다.

실례 12-52

```
(The Script)
#!/bin/bash
1.  for I in 7 9 2 3 4 5
2.  do
     echo $I
3.  done | sort –n
(The Output)
   2
   3
   4
   5
   7
   9
```

설명

1. 정렬되지 않은 수들의 목록에 대하여 for순환을 진행한다.
2. 순환본체에서 수자들을 인쇄한다. 이 출력을 수값정렬인 UNIX의 sort지령의 입력으로 한다.
3. done예약어다음에 파이프가 만들어 진다. 순환은 자식쉘에서 실행된다.

배경에서의 순환실행 순환을 배경에서 실행시킬수 있다. 프로그람은 순환이 끝나기를 기다리지 않고 계속 실행될수 있다.

실례 12-53

```
(The Script)
     #!/bin/bash
1.  for person in bob jim joe sam
     do
2.     mail $person < memo
3.  done &
```

설명

1. for순환은 단어목록에 있는 매 이름 bob, jim, jie, sam들을 밀기한다. 매 이름들이 변수 person에 대입된다.
2. 순환본체에서 매 사람들에게 mem파일의 내용을 보낸다.
3. done예약어의 끝에 있는 &기호는 순환을 배경방식에서 실행하게 한다. 프로그람은 순환이 실행될 동안 계속 집행된다.

7. IFS와 순환

쉘의 내부마당분리기호(IFS)는 공백, 타브, 행바꾸기와 같다. 이것은 read, set, for

와 같이 단어목록을 해석하는 지령에서 단어(어휘)분리기호로서 쓰인다. 가령 목록에서 서로 다른 분리기호를 쓰려면 사용자가 재설정한다. 값을 변화시키기전에 다른 변수에 IFS의 원래 값을 기억시키는것이 좋다. 그렇게 하면 필요에 따라 그의 기정값으로 돌아가기가 쉽다.

실례 12-54

(The Script)

```
$ cat runit2
     #/bin/bash
     # Script is called runit.
     # IFS is the internal field separator and defaults to
     # spaces, tabs, and newlines.
     # In this script it is changed to a colon.
1.   names=Tom:Dick:Harry:John
2.   oldifs="$IFS"    # save the original value of IFS
3.   IFS=":"
4.   for persons in $names
      do
5.      echo   Hi $persons
        done
6.   IFS="$oldifs"               # reset the IFS to old value
7.   set Jill Jane Jolene       # set positional parameters
8.   for girl in $*
     do
9.      echo Howdy $girl
     done
```

(The Output)

```
     $ runit2
5.  Hi Tom
    Hi Dick
    Hi Harry
    Hi John
9.  Howdy Jill
    Howdy Jane
    Howdy Jolene
```

설명

1. name변수는 문자렬 Tom:Dick:Harry:John으로 설정된다. 매 단어들은 두점으로 분리되여 있다.
2. IFS의 값인 공백을 다른 변수 oldfs에 대입한다. IFS의 값이 공백이므로 그 값을 인용부호안에 넣어 기억한다.
3. IFS에 두점을 대입한다.두점은 단어들을 분리시키는데 쓰인다.
4. 변수를 대입한 다음 두점을 단어들사이의 내부마당분리기호로서 사용하여 매 이름들에 대해 for순환을 진행한다.
5. 단어목록의 매 이름들이 현시된다.

6. IFS는 oldfs에 기억되였던 원래 값으로 다시 대입된다.
7. 위치파라메터들을 설정한다. $1에 Jill을, $2에 Jane을, $3에는 Jolene을 대입한다.
8. $*는 모든 위치파라메터들인 Jill, Jane, Jolene과 같다. for순환은 순환이 반복될 때마다 매 이름들을 girl변수에 대입한다.
9. 파라메터목록에 있는 매 이름들이 현시된다.

제 7 절. 함수

함수들은 AT&T의 UNIX체계 VR2에서 Bourne쉘에 도입되였는데 Bourne Again쉘에서 개선되였다. 함수는 지령 혹은 지령묶음에 대한 이름이다. 함수들은 프로그람을 모듈화함으로써 효률적으로 작성할수 있게 한다. 그것들은 현재쉘의 환경에서 실행된다. 다시말하여 자식프로쎄스는 ls와 같은 실행형프로그람을 집행시킬 때와 같이 파생되지 않는다. 사용자는 함수들을 다른 파일에 기억시키고 필요하면 그것들을 스크립트에 적재하여도 된다. 함수들을 리용할 때의 몇가지 중요한 규칙들은 다음과 같이 찾아 볼수 있다.

1. 쉘은 사용자가 별명, 함수, 내부지령 혹은 디스크에 있는 실행프로그람(또는 스크립트)을 리용하겠는가를 결정한다. 그 순서는 별명, 함수, 내부지령, 실행파일이다.
2. 함수는 그것을 리용하기전에 반드시 정의하여야 한다.
3. 함수는 현재환경에서 실행된다. 함수는 자기를 호출한 스크립트와 변수를 공유한다. 국부변수들을 local함수를 써서 함수내부에 만들수 있다.
4. 함수에서 exit지령을 쓴다면 전체 스크립트에서 탈퇴한다. 함수에서 탈퇴하면 그 함수가 호출되였을 때 스크립트가 벗어 났던 곳으로 돌아 간다
5. 함수에 있는 return명령문은 함수안에서 마지막으로 실행된 지령의 탈퇴상태 또는 주어 진 인수의 값을 돌려 준다.
6. 함수를 내부지령 export -f를 써서 자식쉘로 반출할수 있다.
7. 함수를 목록화하여 정의하는데 declare -f지령을 쓴다. 함수이름을 정확히 목록화 하는데는 declare -F[5]를 리용한다.
8. 변수와 같이 트랩프들은 함수안에서 전역적이다. 스크립트와 스크립트에서 호출된 함수들에서 그것들을 공유한다. 트랩프가 함수에서 정의되면 그것은 스크립트에 의해서도 공유된다. 이렇게 하면 예상치 않던 부차적인 결과들이 생길수 있다.
9. 함수가 다른 파일에 있으면 source나 dot지령으로 현재스크립트에 그 함수들을 적재할수 있다.
10. 함수들은 재귀함수일수도 있다. 즉 함수들은 자기자체를 호출할수 있다. 재귀호출회수에는 제한이 없다.

형식

Function 함수이름 {지령들 ; 지령들;}

---

[5] bash 판본은 2.x 판본에서만 쓴다.

실례 12-55

Function dir { echo "Directories: ";ls –1 | awk '/^d/ {print $NF}';}

설명

예약어 function다음에는 함수이름 dir가 놓인다(때로 빈 괄호가 함수이름뒤에 올수 있지만 필요한것은 아니다.). 대괄호안의 지령들은 dir를 건입력할 때 실행된다. 함수의 목적은 현재작업등록부아래의 보조등록부를 현시하는것이다. 첫번째 대괄호의 앞뒤에는 공백을 주어야 한다.

함수의 지우기 함수를 기억기에서 지우기하려면 unset지령을 사용한다.

형식

Unset-f function_name

함수의 반출 함수들을 반출하여 자식쉘에서 그 함수들을 리용할수 있게끔 한다.

형식

Export-f 함수이름

1. 함수의 인수와 되돌이값

함수는 현재쉘에서 실행되므로 변수들은 함수와 쉘의 량쪽에 놓인다. 함수에서 환경변화는 쉘에도 그대로 전달된다.

인수 위치파라메터를 써서 함수에 인수를 넘길수 있다. 위치파라메터들은 함수에 국한된다. 즉 함수에 대한 인수들은 함수밖에서 리용되는 위치파라메터들에 영향을 주지 않는다(실례 12-56을 참고).

local내부함수 함수에만 국한되며 함수가 끝난 다음 없어 지게 되는 국부변수들을 만들려면 local함수를 쓴다(실례 12-57을 참고).

return내부함수 return지령은 함수에서 탈퇴하여 함수를 호출한 곳에서 프로그람에 조종을 돌려 줄 때 쓴다(함수내부를 포함한 스크립트의 임의의 위치에서 exit지령을 쓰면 스크립트가 끝나게 된다.). 함수의 되돌이값은 사용자가 return지령에 특정한 인수로 주지 않는 경우에는 스크립트에 있는 마지막지령의 탈퇴상태값과 같다. return지령에 값을 대입한다면 그 값은 0부터 255사이의 옹근수값만을 가지는 ?변수에 기억된다. return지령의 되돌이값은 0부터 255사이의 옹근수값만을 가지므로 함수의 출력을 얻도록 지령대입을 리용할수 있다. $(function_name)과 같이 화폐기호가 앞에 붙은 괄호안에 또는 거꿀빗선안에 전체 함수를 넣어 UNIX지령의 출력을 얻을 때와 같이 출력을 취하여 변수에 대입한다.

실례 12-56

(Passing Arguments)
(The Script)

```
    #!/bin/bash
    # Scriptname: checker
    # Purpose: Demonstrate function and arguments
1.  function Usage { echo "error: $*" 2>&1; exit 1; }
2.  if (( $# != 2 ))
     then
3.     Usage "$0: requires two arguments"
     fi
4.  if [[ ! (-r $1 && -w $1) ]]
     then
5.     Usage "$1: not readable and writeable"
       fi
6.  echo The arguments are: $*
     <   Program continues here >
```

```
(Output)
$ checker
error:checker:requirs two arguments

$ checker file1 file2
error:file1:not readable and writeable

$ checker filex file2
The arguments are filex file2
```

설명

1. Usage라는 함수를 정의한다. 표준오유(현시장치)에 오유통보문을 내보내는데 쓴다. 함수의 인수들은 함수를 호출할 때 내보내게 되는 여러개의 물음표들로 되여 있다. 그 인수들은 모든 위치파라메터들을 포함하고 있는 특정한 변수인 $*에 기억된다. 함수안의 파라메터들은 국부적이며 함수밖에서 쓰이는 위치파라메터들에 영향을 주지 않는다.
2. 지령행에서 스크립트에 넘어 가는 인수가 2개가 아니면 프로그람은 3행으로 넘어 간다.
3. Usage함수를 호출할 때 문자렬 $0:requires two arguments가 함수에 넘어 가며 $*변수에 기억된다. echo명령문은 다음통보문을 표준오유에로 내보내고 스크립트는 오유가 있다는것을 알려 주는 탈퇴상태 1을 가지고 탈퇴한다.[ㄱ]

4,5. 지령행에서 프로그람으로 넘어 가는 첫번째 인수가 읽기 쓰기가능한 파일의 이름이 아니라면 Usage함수는 $1:not readable and writeable을 인수로 하여 호출된다.

6. 지령행에서 스크립트에 넘어 가는 인수들은 $*에 기억된다. 이것은 함수에서 $*에 영향을 주지 않는다.

ㄱ. 낡은 test형식에서는 식을 if [! \ (-r $1 -a -w $1 \ )]로 쓴다.

실례 12-57

(Using the return Command)

(The Script)

```
$ cat do_increment
     #<!/bin/bash
     # Scnptname. do_increment
1.  increment () {
2.       local sum;       # sum is known only in this function
3.       let "sum=$l + 1"
4.       return $sum      # Return the value of sum to the script.
     }
5.  echo  -n "The sum is "
6.  increment 5      # Call function increment; pass 5 as a
                     # parameter  5 becomes $1 for the increment
                     # function.
7.  echo $?          # The return value is stored, in $?
8.  echo  $sum       # The variable "sum" is not known here
```

(The Output)

```
$ do_increment
4, 6 The sum is 6
8
```

설명

1. increment함수를 정의한다.
2. local내부함수는 sum국부변수를 정의한다. 이 변수는 함수밖에서 존재하지 않는다. 즉 함수에서 탈퇴하면 삭제된다.
3. 함수가 호출되면 함수의 첫 인수 $1의 값은 하나 증가되며 되돌이값은 sum에 설정된다.
4. return내부지령은 인수가 주어질 때 함수가 호출되였던 행다음에 있는 기본 스크립트에로 돌려 보낸다. 그 인수를 ?변수에 기억한다.
5. 현시장치에 문자렬을 출력한다.
6. increment함수는 인수를 5로 하여 호출된다.
7. 함수가 되돌이하면 그의 탈퇴상태는 ?변수에 기억된다. return명령문에서 지정된 인수가 리용되지 않으면 탈퇴상태는 함수에서 실행된 제일 마지막지령의 탈퇴값이다. return에 대한 인수는 반드시 0부터 255사이의 값이여야 한다.
8. sum이 함수 increment에서 정의되였지만 국부변수이므로 함수밖에 정의되지 않는다. 아무것도 현시되지 않는다.

실례 12-58

(Using Command Substitution)

(The Script)

```
$ cat do_square
    #!/bin/bash
    # Scriptname: do_sguare
1.  function square {
        local sq        #SQ is local to the function
        let "sq=$l * $1"
        echo "Number to be squared is $1."
2.      echo "The result is $sq "

            }

3.  echo "Give me a number to square. "

        read number

4.  value_returned=$(square $number) # Command substitution
5.  echo    "$value_returned"
```

(The Output)

```
    $ do_square
3.  Give me a. number to square.
    10
5.  Number to be squared is 10
            The result is 100
```

설명

1. square함수를 정의한다. 사명은 호출될 때 인수 $1을 두제곱하는것이다.
2. 수를 두제곱한 결과가 인쇄된다.
3. 사용자입력을 요구한다.
4. square함수는 사용자가 입력한 수를 인수로 하여 호출된다. 함수가 $가 앞에 붙은 괄호안에 있기때문에 지령바꾸어넣기가 진행된다. 2개의 echo명령문들인 함수의 출력을 변수 value_returned에 대입한다.
5. 지령바꾸어넣기로부터 귀환된 값이 인쇄된다.

2. 함수들과 source(혹은 dot)지령

함수의 기억 등록가입할 때 함수가 정의되도록 하기 위해 보통 함수들을 .profile 파일에서 정의한다. 함수들은 파일에 넘길수도 있고 기억시킬수도 있다. 함수가 필요할 때 source 혹은 dot지령을 파일이름과 함께 리용하여 거기서 함수가 정의되도록 한다.

실례 12-59

```
1   $ cat my functions
2   function go() {
cd $HOME/bin/prog

PS1=' 'pwd' > '

ls

  }

3    function greetings() { echo "Hi $1! Welcome to my world.”; }

4   $ source myfunctions
5   $ greetings george
     Hi george! Welcome to my world.
```

설명

1. myfunctions파일이 현시된다. 2개의 함수는 함수정의를 가지고 있다.
2. 첫번째로 정의된 함수는 go이다. 이 함수는 1차재촉문을 현재작업등록부로 설정한다.
3. 두번째로 정의된 함수는 greetings이다. 이 함수는 인수로 넘겨 진 사용자의 이름을 받는다.
4. source 또는 dot지령은 myfunction의 내용을 쉘기억기에 적재한다. 이때 2개의 함수가 이 쉘을 위해 정의된다.
5. greetings함수가 호출되여 실행된다.

실례 12-60

(The *dbfunctions* file shown below contains functions to be used by the main program. See cd for complete script.)

```
1.  $ cat dbfunctions
2.  function addon () {   # Function defined in file dbfunctions
3.        While true
          do
    echo "Adding information "
    echo "Type the full name of employee "
       read name
    echo "Type address for employee "
    read address
    echo "Type start date for employee (4/10/88 ) :"
    read startdate
    echo $name:$address:$startdate
```

```
echo -n "Is this correct? "
read ans
case "$ans"   in
[Yy]*)
echo "Adding info...'
echo $name:$address $startdate»datafile
sort -u datafile -o datafile
echo -n "Do you want to go back to the main menu?
read ans
if [[ $ans ==[Yy] ]]
then
4.          return        # Return to calling program
        else
5.              continue    # Go to the top of the loop
      fi
            ; ;
      *)
      echo "Do you want to try again? "
      read answer
      case "$answer" in
      [Yy]*) continue; ;
    *) exit; ;
    esac
    ; ;
    esac
    done
6. }    # End   of function definition
 (The Command Line)
7.  $ more mainprog
#'/bin/bash
# Scriptname: mainprog'
# This is the main script that will call the function, addon

datafile=$HOME/bourne/datafile
8.  source dbfunctions    # The file is loaded into memory
      if {!' -e $datafile }
    then
    echo "$(basename $datafile) does not exist" >&2
```

```
        exit 1

        fi
9.        echo "Select one:"
        cat <<EOF
        [1]. Add info
        [2]. Delete   info
        [3]. Update   info
        [4]. Exit
           EOF
        case $choice in
10.       1) addon           # Calling the addon function
        ;;
           2) delete          # Calling the delete function
        ;;
           3)update
        ;;
           4)
        echo Bye
        exit 0
        ;;
        *)   echo Bad choice
        exit 2
        ;;
        esac
        echo Returned from function call
        echo The name is $name
        # Variable set in the function are known in this shell.
        done
```

설명

1. dbfunctions파일이 현시된다.
2. addon함수가 정의된다. 사명은 datafile에 새로운 정보를 추가하는것이다.
3. while순환이 시작된다. break나 continue과 같은 순환조종명령문이 순환본체에 없으면 순환은 계속된다.
4. return지령은 조종을 함수를 호출한 호출프로그람에로 돌려 보낸다.
5. 조종은 while순환의 꼭대기로 돌아 간다.
6. 닫는 대괄호는 순환의 끝을 의미한다.
7. 이부분이 기본스크립트이다. addon함수가 스크립트에서 리용된다.
8. source지령은 dbfunctions파일을 프로그람기억기에 적재한다. 함수 addon이 정의되고 리용가능하게 된다. 이것은 마치도 스크립트의 바로 이 부분에서 함수를 정의한것과 같다.
9. here document와 함께 차림표가 현시된다. 사용자에게 차림표항목을 선택할것을 요구한다.
10. addon함수가 호출된다.

제 8 절. 트랩프신호

프로그람이 실행될 때 Control-C 또는 Control-\ 를 누르면 그 신호가 도착하자마자 프로그람이 끝난다. 신호가 도착하자마자 즉시에 끝내지 않도록 할 필요가 있다. 신호를 무시하여 프로그람이 계속 실행되게 하던가 또는 스크립트에서 실제로 탈퇴하기전에 어떤 정돈조작을 하게 할수 있다. trap 지령은 프로그람이 신호를 받을 때 동작하는 방식을 조종한다. 신호는 한 프로쎄스에서 다른 프로쎄스로 또는 어떤건이 눌리우거나 례외적인 사건이 발생할 때 조작체계에 의해 어떤 프로쎄스로 전송될수 있는 수자로 이루어진 비동기통보문이다.[6] trap지령은 신호를 받으면 쉘에게 현재 실행중에 있는 지령을 끝내라고 알려 준다. trap지령뒤에 인용부호안에 있는 지령들이 놓이게 되면 지정된 신호를 받은 상태에서 지령문자렬이 실행된다. 쉘은 지령문자렬을 두번 즉 trap가 설정될 때와 신호가 도착할 때 읽는다. 지령문자렬이 2중인용부호안에 있으면 모든 변수와 지령바꿔넣기는 trap가 설정될 때 먼저 진행된다.

지령문자렬이 단일인용부호안에 있다면 신호가 검출되여 trap가 실행될 때까지 변수와 지령바꿔넣기는 진행되지 않는다.

지령 kill-l 또는 trap-l을 리용하여 모든 신호들의 목록을 얻는다. 표 12-6에 신호변수들의 목록과 그에 대응한 이름들을 보여 주었다.

형식

```
trap 'command;command' signal_number
trap 'command;command' signal_name
```

실례 12-61

```
trap 'rm tmp*; exit 1-   0 1 2 15
trap 'rm tmp*; exit 1'   EXIT HUP IHT TERM
```

설명

신호 1(정지), 2(새치기), 15(프로그람 끝)가운데서 임의의 신호가 도착하면 모든 파일을 삭제하고 탈퇴한다.

스크립트실행중에 새치기가 들어 오면 trap지령은 사용자가 여러가지 방법으로 새치기신호를 조종하도록 한다. 신호가 도착할 때 그것이 표준으로(기정으로) 처리되게 하던가 무시되게 하던가 또는 호출되는 처리함수(handlerfunction)를 만들수 있다. HUP와 INT와 같은 신호이름들앞에 보통 SIG가 붙는다. 실례로 SIGHUP, SIGINT 등과 같다.[7] bash쉘에서는 SIG앞붙이를 가지지 않는 신호에 대해 기호이름을 리용하거나 신호들에 대해 수값을 리용할수 있게 한다. 쉘이 탈퇴할 때 허위신호이름 EXIT 또는 수 0은 트랩프가 실행되게 한다. 표 12-6에서 보여 준다.

---

6. Bolsky, Morrids I Korn, David G, *The New KornShell Command and Programming* Language (Englewwod Cliffs, NJ: Prenntice Hall PTR, 1995), p. 327

7. 《완전정지》라고도 하는 번호가 9인 SIGKILL은 트랩프되지 않는다.

표 12-6. 신호값과 신호(kill-1)

1)SIGHUP	9)SIGKIU	18)SIGCONT	26)SIGVATALRH
2)SIGINT	10)SIGUSR1	19)SIGSTOP	27)SIGPROF
3)SIGQUIT	11)SIGSEGV	20)SIGSTP	28)SIGWINCH
4)SIGILL	12)SIGUSR2	21)SIGTTIN	29)SIGIO
5)SIGTRAP	13)SIGPIPE	22)SIGTTON	30)SIGPWR
6)SIGANRT	14)SIGALRM	23)SIGURG	
7)SIGBUR	15)SIGTERM	24)SIGXCPU	
8)SIGFPE	17)SIGCHLD	25)SIGXFSZ	

신호의 재설정 신호를 기정방식으로 재설정하려면 trap지령뒤에 신호나 수를 덧붙이면 된다. 함수에서 설정된 trap는 함수가 호출되면 그 함수를 호출한 쉘에 의해 인식된다. 함수밖에서 설정된 trap들도 함수와 같이 인식된다.

실례 12-62

Trap 2 or trap INT

설명

신호 2 즉 SIGINT에 대한 기정동작으로 재설정된다. 기정동작은 Control-C가 눌리워 질 때 프로쎄스를 끝낸다.

신호의 무시 trap지령의 뒤에 빈 괄호가 놓이면 표시된 신호는 프로쎄스에 의해 무시된다.

실례 12-63

Trap 〃〃1 2 or **trap 〃〃 HUP INT**

설명

신호 1(SIGHUP)과 2(SIGINT)는 쉘프로쎄스에 의해 무시된다.

Trap의 현시 모든 트랩프와 그에 대입된 지령을 현시하려면 trap를 입력한다.

실례 12-64

```
(At the command line)
1.  $ trap 'echo "Caught ya!; exit"' 2
2.  $ trap
     trap -- 'echo "Caught ya!; exit 1"' SIGINT
3.  $ trap -
```

설명

1. trap지령은 신호2(Control-C)를 받을 때 탈퇴하도록 설정된다.

2. 인수가 없는 trap지령은 모든 설정된 트랩프들을 현시한다.
3. 인수가 횡선이면 모든 신호들이 초기값 즉 쉘이 기동하였을 때의 값들로 재설정된다.

실례 12-65

(The Script)

```
  # !/bin/bash
  # Scriptname: trapping
  # Script to illustrate the trap command and signals
  # Can use the signal numbers or bash abbreviations seen
  # below. Cannot use SIGNIT, SIGQUIT,etc.
1. trap 'echo "Control-C will not terminate $0." ' INT
2. trap 'echo "Control-\  will not terminate $0." ' QUIT
3. trap 'echo "Control-z will not terminate $0." ' TSTP
4. echo  "Enter any string after the prompt.
    When you are ready to exit, type \ "stop\ "."
5. while true
    do
6. echo –n "Go ahead...> "
7. read
8. if [[$REPLY== [Ss]top ]]
    then
9. break
    fi
10.  done
```

(The Output)

```
$ trapping
4. Enter any string after the prompt.
    When you are ready to exit, type "stop"
6. Go ahead...> this is it^C
1. Coatrol-C will not terminate trapping.
6. Go ahead...> this is it again^Z
3. Control-Z will not terminate trapping.
6. Go ahead... > this is never it|^ \
2. Control-\  will not terminate trapping.
6. Go ahead...> stop
    $
```

설명

1. 첫번째 trap는 Control-C인 INT신호를 받는다. 프로그람이 동작할 때 Control-C를 누르면 괄호안에 있는 지령이 실행된다. 프로그람은 중지되지 않고 Control-C will not terminate traping이라는 통보문을 출력하고 사용자입력을 대기한다.

2. 두번째 trap는 사용자가 Control-\ 을 출력했을 때 즉 QUIT신호인 Control-\ 을 누를 때 실행된다. 문자렬 Control-\ will not terminate traspping이 현시되고 프로그람이 계속 실행된다. 기정으로 SIGQUIT인 신호는 프로쎄스를 끌내고 core파일을 작성한다.
3. 세번째 trap는 사용자가 TSTP신호인 Control-Z 를 누를 때 실행된다. 문자렬 Control-Z will n ot terminate trapping이 현시되고 프로그람이 계속 실행된다. 보통 이 신호는 일감조종이 실행되면 프로그람이 배경에서 중단되게 한다.
4. 사용자입력을 요구한다.
5. while순환이 진행된다.
6. 문자렬 Go ahead...>가 인쇄되고 프로그람은 입력을 대기한다.
7. read지령은 내부변수 REPLY에 사용자입력값을 대입한다.
8. REPLY값이 Stop 혹은 stop라면 break지령에 의해 순환에서 탈퇴하고 프로그람을 끌낸다. Stop나 stop를 입력해야만 이 프로그람에서 탈퇴할수 있다. 그렇지 않으면 kill지령을 리용하여 탈퇴할수 있다.
9. break 지령은 순환본체에서 탈퇴하게 한다.
10. done예약어는 순환의 끝을 표시한다.

신호의 재설정 신호를 그의 기정방식으로 재설정하려면 trap지령뒤에 신호이름이나 수자를 붙여야 한다.

실례 12-66

Trap 2

설명

프로쎄스를 끌내는데 리용되는 Control-C인 신호 2 즉 SIGNIT에 대한 기정동작을 재설정한다.

실례 12-67

Trap ‘trap 2 ‘ 2

설명

신호가 도착할 때 인용부호안에 있는 지령문자렬을 실행하도록 신호 2(SIGNIT)에 대한 기정동작을 설정한다. 사용자는 프로그람을 끌내기 위해 Control-C를 두번 눌러야 한다. 첫번째 trap는 신호를 받고 두번째 trap는 프로쎄스를 끌내는 기정동작으로 트랩프를 재설정한다.

함수안에서 트랩프 사용자가 트랩프를 리용하여 함수에서 신호를 처리하면 함수가 호출될 때 그것은 전체 스크립트에 영향을 준다. 트랩프는 스크립트에 대해여 전역적이다. 다음 실례에서 새치기건 ^C를 무시하도록 설정하였다. 이 스크립트에서 순환을 중지하려면 kill지령을 리용하여 한다.

실례 12-68

(The Script)

```
#!/bin/bash
1.  function trapper () {
        echo "In trapper"
2.      trap 'echo "Caught in a trap!"' INT
     # Once set, this trap affects the entire script. Anytime
     # ^C is entered, the script will ignore it.
     }
3.  while :
     do
         echo "In the main script"
4.       trapper
5.       echo "Still in main"
         sleep 5
  done
   (The Output)
   $ trapper
   In the main script
   In trapper
   Still in main
   ^CCaught in a trap!
   In the main script
```

설명

1. trapper함수가 정의된다. 이 함수에서 모든 변수와 트랩프들은 스크립트에 대하여 전역적이다.
2. trap지령은 신호 2 즉 INT(^c)를 무시한다. ^c가 눌러 지면 Caught in a trap통보문이 출력되며 스크립트는 계속 실행된다. 스크립트는 kill지령이나 Control-\ 로 중지시킬수 있다.
3. 기본스크립트는 무한순환을 시작한다.
4. trapper함수를 호출한다.
5. 함수가 복귀되면 이 행에서 실행이 시작된다.

제 9 절. 오유수정

bash지령에 –n추가선택을 리용하면 지령들을 실행시켜 보지 않고도 스크립트물음표검사를 할수 있다. 즉 스크립트에 문법오유가 있으면 쉘이 오유를 알려 준다. 오유가 없으면 아무것도 현시하지 않는다. 스크립트를 오유수정하는데 널리 리용되는 방법은 –x추가선택을 가진 set지령과 –x추가선택과 스크립트이름으로 호출되는 bash이다. 표 12-7에 오유수정추가선택들을 보여 주었다. 이 추가선택들은 스크립트의 실행을 추적할수 있게 한다. 스크립트의 매 지령은 바꿔넣기가 진행된후 현시되며 그다음에 실행된다. verbose추가선택이 선택되거나 –v추가선택(bash-v스크립트)으로 쉘을 호출하면스크립트의 매행이 입력된 그대로 현시되고 그다음에 실행된다. 스크립트의 행이 현시될 때 그 앞에 더하기(+)기호가 붙는다.

표 12-7. debug추가선택

지령	추가선택	사명
bash-x scriptname	Echo option	스크립트의 매개 행을 변수바꿔넣기한 후 실행전에 현시한다.
bash-v scriptname	Verbase option	스크립트의 매개 행을 입력된대로 실행전에 현시한다.
bash-n scriptname	Noexec option	지령을 해석하지만 실행하지 않는다.
set –x	Terns on echo	스크립트에서 실행을 추적한다.
set +x	Terns off echo	추적을 해제한다.

실례 12-69

```
(The script)
$ cat todebug
    #!/bin/bash
    # Scriptname: todebug
1.  name="Joe Shmoe"
    if [[ $name == "Joe Blow" ]]
    then
        printf "Hellow $name\ n"
    fi
    declare –i num=1
    while (( num<5 ))
    do
        let num+=1
    done
    printf  "The total is  %d\ ", $num

(The Output)
2.  bash –x  todebug
    + name=Joe Shmoe
   +[[JoeShmoe== \ J\ o\ e\ \ B\ l\ o\ w]]
   + declare –i num=1
   + ((  num<5  ))
   + let num+=1
   + ((  num<5  ))
   + let num+=1
   + ((  num<5  ))
   + let num+=1
   + ((  num<5  ))
   + let num+=1
   + ((  num<5  ))
   +printf 'The total is %d\ n, ' 5
The total is 5
```

설명

1. 스크립트의 이름은 todebug이다. -x추가선택을 설정하여 스크립트가 실행하는것을 볼수 있다. 매 순환에서 프로쎄스가 현시되며 변수들의 값이 설정되거나 변경될 때 인쇄된다.
2. bash가 –x추가선택으로 호출된다. echoing이 설정된다. 스크립트의 매 행의 앞에 더하기(+)기호가 붙여 져 현시장치에 현시된다.

제 10 절. getopts에 의한 지령행추가선택처리

지령행추가선택들을 요구하는 스크립트를 작성할 때 위치파라메터가 항상 효과적인것은 아니다. 실례로 UNIX ls지령에는 많은 지령행추가선택과 인수들이 있다(추가선택들은 횡선을 요구하지만 인수는 요구하지 않는다.). 추가선택은 ls-laFi, ls-i-a-l-f, ls-ia-F등과 같이 여러가지 방법으로 프로그람에 넘길수 있다. 인수들을 요구하는 스크립트가 있다면 위치파라메터들은 ls-l-i-F와 같이 인수들을 개별적으로 처리하는데 리용된다. 매 횡선추가선택들은 각각 $1, $2, $3에 기억된다. 그런데 만일 사용자가 ls-liF에서와 같이 한개의 횡선으로 모든 추가선택들을 표시했다면 어떻게 되겠는가? 이때 –liF는 모두 스크립트에서 $1에 대입된다. getops함수는 지령행추가선택들과 인수들을 ls프로그람에 의해 처리되는것과 같은 방식으로 처리할수 있게 한다.[8] getops함수는 여러가지 조합을 사용하여 runit프로그람이 인수들을 처리하게 한다.

실례 12-70

(The Command Line)
1. $ **runit -x -n 200 filex**
2. $ **runit -xn200 filex**
3. $ runit **-xy**
4. $ runit **-yx** -n 30
5. $ **runit -n250 -xy filey**

(any other combination of these arguments)

설명

1. runit프로그람은 4개의 인수를 가진다. x는 추가선택, n은 그다음 수값인수를 요구하는 추가선택, filex는 독립적으로 있는 인수이다.
2. runit프로그람은 추가선택 x, n, 수값인수 200을 조합한다. filex는 역시 인수로 된다.
3. runit프로그람은 조합된 x와 y추가선택에 의하여 호출된다.
4. 조합된 x와 y추가선택에 의하여 runit프로그람이 호출된다. n추가선택은 수인수 30과 같이 따로 넘겨 진다.
5. n추가선택과 수인수가 조합되여 runit프로그람이 호출되며 x와 y추가선택이 조합되며 filey는 따로 분리한다.

runit프로그람에 대해 자세히 학습하기전에 getops를 사용하는 프로그람으로부터 행을 검사하여 그것이 인수들을 어떻게 처리하는가를 본다.

---

8. C서고함수 getops에 대해서는 UNIX의 사용지도서페지(3절)를 참고하시오.

실례 12-71

(A Line from the Script Called “runit”)
while getopts :xyn: name

설명

1. x, y, n은 추가선택들이다. 이 실례에서 첫번째 추가선택앞에 옹근두점이 있다. 이것은 getopts가 오유를 통보하지 않게 한다. 만일 추가선택들중 어느한개의 뒤에 옹근두점이 있으면 그 추가선택뒤에 공백으로 분리된 인수가 놓인다. 인수는 횡선으로 시작하지 않는 단어이다. −n추가선택은 인수를 요구한다.
2. 지령행에 입력된 임의의 추가선택들은 횡선으로 시작되여야 한다.
3. 횡선이 없는 임의의 추가선택들은 getops에게 추가선택목록이 끝이라는것을 알려 준다.
4. getopts가 호출될 때마다 자기가 찾는 다음 추가선택값을 변수 name에 넣는다(변수이름은 임의로 할수 있다.). 만일 비법인수가 주어 지면 name에 물음표가 대입된다.

getopts스크립트 다음의 실례들은 getopts가 어떻게 인수를 처리하는가를 보여 준다.

실례 12-72

```
(The  Script)
  $ cat  opts1
  #!/bin/bash
  # Program: opksl
  # Using getopks -- First try –
1.  while getopts xy options
     do
2.     case $options in
3.     x) echo "you entered -x as an option"
       y) echo "you entered -y as an option"

       esac
     done
(The Command Line)
4.  $ opts1 –x
     you entered -x as an option
5.  $ opts1 –xy
     you entered -x as an option
     you entered -y as an option
6.  $ opts1–y
     you entered -y as an option
7.  $ opts1 –b
     optsl:   illegal option  -- b
8.  $ opts1b
```

설명

1. getopts지령은 while지령의 조건으로 리용된다. 이 프로그람에 대한 유효추가선택들이 getops지령뒤에 표시된 x와 y이다. 매 추가선택은 순환본체에서 하나씩 검사되며 횡선이 없이 변수 option에 대입된다. 더이상 처리할 인수가 없을 때 getops는 령이 아닌 상태로 탈퇴하여 while순환을 끌나게 한다.
2. case지령은 option변수에 있는 모든 가능한 추가선택들인 x 또는 y를 검사하는데 리용된다.
3. x가 추가선택이였다면 문자렬 You entered x as an option을 현시한다.
4. 지령행에서 opts1스크립트에 x추가선택이 주어 진다.
5. 지령행에서 opts1스크립트에 xy추가선택이 주어 진다. x와 y는 getops에 의해 처리되는 합법적추가선택들이다.
6. 지령행에서 opts1스크립트에 y추가선택을 주면 getops에 의해 처리되는 하합법적인 추가선택이다.
7. opts1스크립트에 비합법적추가선택인 b추가선택이 주어 진다. getopts는 표준오유통보문 stderr를 보낸다.
8. 앞에 횡선이 붙지 않은 추가선택은 추가선택으로 되지 않고 getopts가 인수처리를 정지하게 한다.

실례 12-73

(The Script)

```
   $ cat opts2
   #!/bin/bash
   # Program: opts2
   # Using gekopts — Second try --
1.  while getopts xy options 2> /dev/null
    do
2.        case $options in
           x) echo "you entered -x as an option";;
           y) echo "you entered -y as an option";;
3.        \ ?) echo "Only -x and -y are valid options"   1>&2;,
           esac
    done
```

(The Command Line)

```
$ opts2 –x
you entered -x as an option
$ opts2 –y
you entered -y as an option
$ opts2 xy
$ opts2 -xy
you entered -x a.ss an option
```

you entered -y as an option

4 $ **opts2 -g**
Only -x and -y are valid options

5. $ **opts2 -c**
Only -x and -y are valid options

설명

1. getopts로 부터 오유통보가 생기면 그것은 /dev/null로 방향바꾸기된다.
2. 추가선택이 비합법추가선택이면 물음표를 options변수에 대입한다. case지령은 물음표를 검사하는데 리용되여 오유통보문을 표준오유로 인쇄할수 있게 한다.
3. options변수에 물음표를 대입하면 case명령이 실행된다. 물음표가 거꿀빗선으로 보호되여 쉘은 그것을 통용기호로 보지 않고 파일이름바꿔넣기를 진행하려고 한다.
4. g는 합법적추가선택이다. 물음표가 변수 options에 대입되고 오유통보문을 현시한다.
5. C는 합법적추가선택이 아니다. 물음표가 변수 options에 대입되고 오유통보문을 현시한다.

특수 getopts변수 getopts함수는 2개의 변수를 제공하여 인수들을 추적할수 있게 한다. 그 변수들은 OPTIND, OPTARG이다. OPTIND는 1로 초기화되여 getopts가 지령행인수처리를 끝낼 때마다 다음번에 처리될 인수의 번호로 증가되는 특수변수이다. OPTARG변수에는 합법적인수의 값이 들어 있다(실례 12-74와 12-75에서 보여 준다.).

실례 12-74

(The Script)

```
   $ cat opts 3
   #!/bin/bash
   # Program: opts 3
   # Using getopts — Third try —
1.  while getopts dq: options
    do
       case $options in
2.            d) echo "-d is a valid switch ";;
3.            q) echo "The argument for -q is $OPTARG"; ;
               \ ?) echo "Usage:opts3 -dq filename ... " 1>&2;;
               esac
         done
```

(The Command Line)

4 $ **opts3 -d**
-d is *a. valid switch*

5 $ **opts3 -q foo**

The argument tor -q is foo

6 $ **opts3 -q**
Usage:opts3 -dq filename ...

7 $ **opts3 -e**
Usage:opts3 -dq filename ...

8 $ **opts3 e**

설명

1. while지령은 getopts의 탈퇴상태를 검사한다. getopts가 인수를 성과적으로 처리하면 그것은 령탈퇴상태로 되돌리고 while순환이 시작된다. 인수목록에 붙은 웅근두점은 q추가선택이 인수를 요구한다는것을 의미한다. 인수는 특수변수 OPTARG에 기억된다.
2. 합법적추가선택의 다른 하나는 d이다. d가 추가선택으로 입력되면 d(횡선이 없는)는 option변수에 기억된다.
3. q는 합법적추가선택이다. 이 추가선택은 인수를 요구한다. q추가선택과 그 인수사이에 공백이 있어야 한다. 만일 q가 그뒤에 인수가 오는 추가선택으로 입력되면 횡선이 없이 options변수에 기억되고 인수는 OPTARG변수에 기억된다. 이 추가선택뒤에 인수가 없으면 물음표가 변수 options에 기억된다.
4. d추가선택은 opts3에 대한 합법적추가선택이다.
5. 인수가 있는 q추가선택은 opts3에 대해서도 합법적추가선택이다.
6. 인수가 없는 추가선택 q는 오유로 된다.
7. e추가선택은 무효추가선택이다.추가선택이 비합법적이면 물음표가 options변수에 기억된다.
8. 추가선택앞의 횡선도 더하기부호가 붙지 않는다. getopts지령은 그것을 추가선택으로 처리하지 않고 0이 아닌 탈퇴상태를 복귀한다. while순환이 끝난다.

실례 12-75

```
$ cat opts4
# bin bash
# Program: opts4
# Using gekopts — Fourth try —
1.  while getopts xyz: arguments 2>/dev/null
     do
     case $arguments   in
2.        x) echo "you entered -x as an option ."; ;
          y) echo "you entered -y as an option." ;;
```

```
3.        z) echo “you entered –z as an option.”
          echo "\  $OPTARG is $OPTARG.”;;
4.   \  ?)echo "Usage opts4 [-xy] [-z argument]"
             exit 1;;
          esac
       done
5.   echo “The number of argument passed was $(( $OPTIND –1))
```

(The Cointnand Line) $ **opts4 -xyz**
foo

You entered -x as an *option.*

You entered -y as an *option.*

You entered, -z as an option.

$OPTARG is foo.

The number of arguments passed was 2.

$ **opts4 –x –y –z boo**

You entered -x as an *option.*

You entered -y as an *option.*

You entered, -z as an option.

$OPTARG is boo.

$ **opt4 -d**

Usage: opt4 [-xy] [-z argument]

설명

1. while지령은 getopts의 탈퇴상태를 검사한다. getopts가 인수를 성과적으로 처리하면 령탈퇴상태를 되돌리고 while순환이 시작된다. z추가선택에 붙은 옹근두점은 getops에게 –z추가선택뒤에 인수가 있어야 한다는것을 알려 준다. 추가선택이 인수를 가지면 인수 getops는 내부변수 OPTARG에 기억된다.
2. x가 추가선택으로 주어 지면 변수 argament에 기억된다.
3. z가 인수가 있는 추가선택으로 주어 지면 그 인수는 내부변수 OPTARG에 기억된다.
4. 무효추가선택이 입력되면 물음표가 변수 argument에 기억되고 오유통보를 현시한다.
5. 특수 getopts변수 OPTIND는 다음번에 처리될 추가선택의 번호를 가진다. 그 값은 언제나 지령행인수들의 실지번호보다 하나 더 크다.

제 11 절. eval지령과 지령행의 해석

eval지령은 지령행을 평가하고 모든 쉘바꿔넣기를 진행한 다음에 지령행을 실행한다. 지령행에 대한 표준구문해석이 사용자의 요구를 충분히 만족시키지 못할 때 이것을 리용한다.

실례 12-76

1. $ **set a b c d**
2. $ **echo The last argument is \ $$#**
   The last argument is $4
3. $ **eval echo The last argument is \ $$#**
   The last argument is d
4. $ **set –x**
   $ **eval echo The last argument is \ $$#**
   + eval echo The last argument is '$4'
   ++ echo the last argument is d
   The last argument is d

설명

1. 4개의 파라메터들을 설정한다.
2. 요구되는 결과는 마지막위치파라메터의 값을 인쇄한다. \ $는 화페기호를 문자그대로 인쇄한다. $#는 위치파라메터들의 개수 4와 같다. 쉘은 $#를 평가한후 $4의 값을 얻기 위하여 행을 다시 구문해석하지 않는다.
3. $4가 마지막인수대신 인쇄된다.
4. 쉘이 변수바꿔넣기를 진행한후 eval지령이 변수바꿔넣기를 진행하고 echo 지령을 실행한다.
5. echoing을 설정하여 구문해석순서를 보여 준다.

실례 12-77

(From Shutdown Program)
1. **eval '/usr/bin/id | /usr/bin/sed •s/** ***1^a.-z0-9=} .* I I'***
2. ["${uid:=0}" -ne 0]
   then
3. echo $0:Only root can run &0
   exit 2
   fi

설명

이것은 좀 복잡하다. id 프로그람의 출력을 sed 에 전송하여 문자렬의 uid 부

분을 추출한다. id에 대한 출력은

Uid=9496(elite) gid=40 gromps=40

Uid=0(root) gid=1(daemon) gromps=1(daemon)

sed의 정규식은 다음과 같다.

문자렬의 처음에서 시작하여 문자, 수자 또는 갈기부호가 아닌 임의의 물음표를 찾아서 그 문자와 그뒤에 오는 모든 문자들을 지운다. 그 결과는 첫번째 열기괄호로부터 행의 마지막까지 모든것을 지운것이다. 나머지는 다음과 같다.

Uid=9496

혹은

uid=0

eval은 지령행을 평가한 다음에 그 결과로 생기는 지령을 실행한다.

uid=9496또는 uid=0

실례로 사용자 ID가 root이면 실행되는 지령은 uid=0이다. 이 지령은 쉘에서 uid라는 국부변수를 만들며 그 변수에 0을 대입한다.

2. uid변수의 값이 0인가를 검사한다. 이것은 지령변경자를 리용하여 진행한다.
3. uid가 0이 아니면 echo지령은 스크립트이름($0)과 통보문을 출력한다.

제 12 절. bash추가선택

1. 쉘호출추가선택

bash지령을 리용하여 쉘을 기동시킬 때 그것은 동작방식을 변경시키기 위한 추가선택들을 가질수 있다. 두가지 형태의 추가선택들 즉 단일문자추가선택과 다중문자추가선택들이 있다. 하나는 단일문자추가선택이고 다른 하나는 다중문자추가선택이다. 단일문자추가선택은 한개의 단일문자와 한개의 횡선으로 되여 있다. 다중문자추가선택은 2개의 횡선과 여러개의 물음표들로 이루어 진다. 다중문자추가선택들은 단일문자추가선택보다 앞서 나타나야 한다.

대화형등록가입쉘은 보통 –i(대화형쉘을 기동한다.), –s(표준입력로부터 읽는다.) –m(일감조종을 할수 있게 한다.)으로 시작한다(표 12-8에 보여 준다.).

표 12-8. bash2.x 쉘호출추가선택

추가선택	의 미
-c string	지령은 string로부터 읽어 진다. string뒤에 있는 임의의 인수들이 $0에서 시작하는 위치파라메터들에 대입된다.
-D	$가 앞에 붙은 2중인용부호안에 있는 문자렬들의 목록에 표준출력이 인쇄된다. 현재작업환경이 C나 POSIX가 아니면 이 문자들은 언어를 해석해야 한다. -n추가선택이 리용되면 지령이 실행되지 않는다.

(표계속)

추가선택	의 미
-I	쉘은 대화형방식에 있다. TERM, QUIT, INTERRUPUT는 무시된다.
-s	지령들이 표준입력으로부터 읽어 지고 위치파라메터들이 설정되게 한다.
-r	제한된 쉘을 시동한다.
--	추가선택들의 끝을 알리고 추가선택처리가 더이상 진행되지 않게 한다. -- 또는 –뒤에 있는 임의의 인수들은 파일이름과 인수들로 취급된다.
-dump_strings	-D와 같다.
-help	내부지령에 대한 사용법통보문을 현시하고 탈퇴한다.
--login	bash가 등록가입쉘로 호출되게 한다.
-noediting	bash가 대화형으로 시동할 때 readline서고를 리용하지 않는다.
-noprofile	bash쉘은 시동할 때 초기화파일들인 /etc/profile, ~bash_profile 또는 ~/.bash _login, ~/.profile 을 읽지 않는다.
-norc	Bash는 대화형쉘들에 대하여 ~/.bashrc파일을 읽지 않는다. 쉘을 Sh로 기동한다면 기정값은 on이다.
-posix	bash 의 동작방식이 표준 POSIX 1003.2가 아니면 그에 맞도록 변경시킨다.
-quiet	쉘이 기동할 때 어떤 정보도 현시하지 않는다.
-refile파일	Bash가 대화형이면 ~/.bashrc대신에 이 초기화파일을 리용한다.
-restricted	제한된 쉘이 기동한다.
-version	bash쉘에 대한 판본정보를 현시하고 탈퇴한다.

표 12-9. bash(2x이전)쉘 호출추가선택

추가선택	의 미
-c string	지령들은 string으로부터 읽어 진다. string다음의 임의의 인수는 $0으로 시작하고 위치파라메터에 기억된다.
-D	$로 시작하는 2중인용부호안의 물음표문자렬들이 표시된다. 이 렬들은 현재환경이 c나 posix가 아닐 때 언어해석을 기본으로 한다. -n추가선택은 지령이 실행되지 않음을 의미한다.
-i	쉘은 대화형방식에 있다. TERM, QUIT, INTERRUPUT는 무시된다.
-s	표준입력으로부터 지령읽기, 위치파라메터를 설정하게 한다.
-r	제한된 쉘기동
--	추가선택들의 끝을 알리고 추가선택처리가 더이상 진행되지 않게 한다. -- 또는 –뒤에 있는 임의의 인수들은 파일이름과 인수들로 취급된다.
-login	등록가입쉘로서 bash를 리용한다.

(표계속)

추가선택	의 미
-nobracexpansion	대괄호가 무시된다.
-nolineediting	Bash는 기동할 때 readline서고를 리용하지 않는다.
-noprofile	bash쉘은 시동할 때 초기화파일들인 /etc/profile, ~bash_profile 또는 ~/.bash _login, ~/.profile 을 읽지 않는다.
-posix	Bash의 기동을 posix표준에 맞춘다.
-quiet	쉘이 기동할 때 어떤 정보도 현시하지 않는다.
-rcfile file	Bash가 대화형이면 ~/.bashrc대신 이 초기화파일을 리용한다.
-verbose	Verbose를 설정한다. –v와 같다.
-version	이 bash쉘에 대한 판본정보를 현시하고 탈퇴한다.

2. set지령과 추가선택

set지령을 리용하여 지령행인수들을 처리할 때와 같이 쉘추가선택들을 절환할수 있다. 추가선택을 설정하려면 횡선(-)을 추가선택앞에 붙인다. 추가선택을 해제하려면 추가선택앞에 더하기기호를 붙인다. 표 12-10에서 –set추가선택들의 목록을 보여 준다.

실례 12-78

```
1. $ set -f
2. $ echo *
3. $ echo
   ??
4. $ set +f
```

설명

1. f추가선택이 설정되여 파일이름확장이 진행될수 없게 한다.
2. 별표는 확장되지 않는다.
3. 물음표는 확장되지 않는다.
4. f는 해제된다. 파일이름확장이 가능하다.

표 12-10. set내부지령과 추가선택

추가선택이름	추가선택	기 능
allexpor	-a	추가선택이 설정된 때부터 해제될 때까지 넘길수 있는 새로운 변수 또는 수정된 변수들을 자동적으로 표시한다.
braceexpand	-b	괄호확장을 가능하게 하는데 기정설정이다.
emacs		지령행편집을 위해 emacs내부편집기를 리용하는데 기정설정이다.

(표계속)

추가선택이름	추가선택	기 능
errexit	-e	지령이 0 아닌 탈퇴상태(실패)를 복귀하면 탈퇴된다. 초기화파일을 읽을 때 설정되지 않는다.
histexpan	-H	리력바꿔넣기를 진행할 때 !와 !!를 가능하게 하는데 이것은 기정으로 설정된다.
history		지령행리력을 가능하게 한다. 기정으로 설정된다.
ignoreeof		EOF(Control-D)로 쉘을 탈퇴하지 못하게 한다. exit를 입력해야 한다. 쉘변수설정 IGNOREEO와 같다.
keyword	-k	예약어인수들을 지령에 대한 환경에 놓는다.ㄱ
interactive comments		대화형쉘에 대하여 #를 앞에 붙여 행우의 나머지 본문을 설명한다.
monitor	-m	일감조종을 하게 한다.
noclobber	-c	방향바꾸기가 리용될 때 파일우에 덧씌여 지지 않게 한다.
noexec	-n	지령들을 읽기만 하고 실행하지 않는다. 스크립트들은 물음표법을 검사하는데 리용된다. 실행될 때 설정되지 않는다.
noglob	-d	경로이름완성을 할수 없게 한다. 즉 통용기호를 해제한다.
notify	-b	배경일감이 끝날 때 사용자에게 알려 준다.
nounset	-u	설정되지 않은 변수를 확장할 때 오유를 현시한다.
onecmd	-t	한개의 지령을 읽고 실행한후 탈퇴한다.
physical	-p	cd, pwd를 입력하고 기호련결은 하지 않는다. 물리등록부가 대신 리용된다.
posix		기정동작이 POSIX표준과 맞지 않으면 쉘동작방식을 변화시킨다.
privileged	-p	설정되면 쉘은 .profile이나 ENV파일을 읽지 않으며 쉘함수들은 환경값을 계승하지 않는다. setuid스크립트에 대해 자동적으로 설정된다.
posix		기정동작을 POSIX1003.2로 쉘동작방식을 변화시킨다.
verbose	-v	오유수정을 위해 verbose방식을 설정한다.
vi		지령행편집을 위해 vi내부편집기를 리용한다.
xtrace	-x	오유수정을 위해 echo방식을 설정한다.

ㄱ. 추가선택은 bash2.x 판본에만 적용한다.

3. shopt지령과 추가선택

shopt(bash2.x)지령을 리용하여 쉘추가선택들을 절환할수 있다.

표 12-11. shopt지령추가선택

추가선택	의 미
cdable_vars	cd내부지령의 인수는 등록부가 아니면 그 값을 변화시키려는 등록부변수이름으로 될수 있다.
cdspell	cd지령에서 등록부이름의 맞춤법오유를 정정한다. 검사하는 오유는 문자의 순서바뀜, 빠뜨림, 한개 문자가 너무 많은 오유이다. 정정이 진행되면 정정된 경로가 인쇄되고 지령이 실행된다. 대화형쉘에서만 리용된다.
checkhash	bash는 하쉬표에 있는 지령을 실행하기전에 그것이 존재하는가를 검사한다. 하쉬표에 있는 지령이 더이상 존재하지 않으면 표준경로탐색이 진행된다.
checkwinsize	bash는 매 지령뒤에 있는 창문의 크기를 검사하고 필요하면 LINES와 COLUMS의 값을 변경한다.
cmdhist	bash는 같은 리력입력에 있는 다중행지령의 모든 행들을 기억하려고 한다. 이것은 다중행지령의 재지령을 쉽게 한다.
dotglob	bash는 파일이름확장의 결과에서 점으로 시작하는 파일이름을 포함한다.
execfail	exec내부지령에 대한 인수로서 지정된 파일을 실행시키지 못하면 비대화형쉘은 탈퇴할수 없다. exec가 실패하면 대화형쉘은 탈퇴하지 못한다.
expand_aliases	별명들이 확장된다. 기정으로 줄수 있다.
extglob	확장된 패턴대조특성들(파일이름확장을 위해 Korn쉘로부터 넘어온 정규표현메타문자들)이 가능하다.
histappend	쉘이 탈퇴할 때 HISTFILE이 값으로 이름 지어 진 파일에 리력목록을 추가한다. 이때 파일에 겹쳐쓰기하지는 않는다.
histreedit	readline이 리용되면 사용자는 실패한 리력바꿔넣기를 다시 편집할수 있다.
histverify	리력의 결과를 직접 쉘해석기에 보내지 않고 readline편집기로 보내여 수정하게 한다.
hostcomplete	이것이 설정되고 Readline을 리용할수 있으면 bash는 @를 포함하는 단어가 완성될 때 호스트이름완성을 진행하려 한다. 기정이 가능하다.
huponexit	대화형등록가입쉘에서 탈퇴할 때 모든 job에 SIGHUP신호를 보낸다.
interactive_comments	#로 시작하는 단어는 그 단어와 그 행에 있는 모든 나머지문자들이 대화형쉘에서 무시되게 한다. 기정이 가능하다.
lithist	이것을 리용할수 있고 cmdhist추가선택이 추가선택을 리용할수 있다면 다중행지령들이 반두점분리기를 리용하지 않고 내장된 행바꾸기문자들을 리용하여 리력에 기억된다.
mailwarm	이것이 설정되고 bash가 우편을 위해 검사하는 파일이 그것이 검사된 마지막시점으로부터 호출된적이 없으면 통보 The mail in mailfile has beenread가 현시된다.

(표계속)

추가선택	의 미
nocaseglob	이 추가선택이 설정되면 bash는 파일이름확장을 진행할 때 대소문자구별없이 파일이름을 대조한다.
nullglob	이 추가선택이 설정되면 bash는 파일과 대조하지 않는 파일이름패턴들을 그자체가 아니라 빈문자렬로 확장하게 한다.
Promptvars	이 추가선택이 설정되면 재촉문은 문자렬에서 확장된후에 변수와 파라메터확장이 진행된다.기정으로 가능하다.
Restricted_shell	쉘은 제한된 방식에서 시동되면 이 추가선택들을 설정한다. 그 값은 변화시킬수 없다. 기동파일이 기동될 때 쉘이 제한되는가 제한되지 않는가를 조사하게 한다.
Shift_verbose	이것이 설정되면 shift내부지령은 자리밀기회수가 위치파라메터의 개수를 초과할 때 오유통보문을 인쇄한다. 설정되면 source내부지령은 PATH의 값을 리용하여 인수로 제공된 파일을 포함하는 등록부를 찾는다.기정으로 가능하다.
Source	점(.)과 같은 의미이다.

제 13 절. 내부지령

쉘은 자기의 원천코드에 많은 내부지령들을 가지고 있다. 지령들이 많이 있기때문에 쉘은 디스크에서 지령들을 찾지 않고 실행속도를 높일수 있다. Bash와 함께 제공된 help특성은 임의의 내부지령에 대한 직결방조를 준다. 표 12-12에서 내부지령들을 보여주었다.

표 12-12. 내부지령

지 령	기 능
.	현재프로쎄스의 환경에서 프로그람을 실행한다. source와 같다.
. file	dot지령은 파일로부터 지령을 읽고 실행한다.
:	아무것도 하지 않는 지령이다. 0출력상태를 돌려 준다.
alias	존재하는 지령에 대한 《별명》을 표시하고 작성한다.
bg	일감을 배경에 넣는다.
bind	현재건과 기능맺기를 현시한다. 또는 건을 readline함수나 마크로와 맺어 준다.
break	맨안쪽 순환에서 탈퇴한다.
break[n]	706페지의 《break지령》을 참고
builtin[sh_builtin[ags]]	쉘의 내부지령을 인수로 넘겨 실행하고 령탈퇴상태를 되돌린다. 함수와 내부지령이 같은 이름을 가질 때 유효하다.
cd[arg]	인수가 없으면 등록부를 홈으로 변경시키고 인수가 있으면 등록부를 인수의 값으로 변경시킨다.

(표계속)

지 령	기 능
command command[arg]	함수가 같은 이름을 가질 때에도 지령을 실행시킨다. 즉 함수탐색을 무시한다.
continue [n]	707페지의 《continue지령》을 참고
declare[var]	모든 변수들을 표시하거나 선택적속성들을 가진 변수들을 선언한다.
dirs	pushd로부터 생기는 현재 기억된 등록부의 목록을 현시한다.
disown	일감표로부터 능동일감을 삭제한다.
echo[args]	행바꾸기로 끝나는 인수들을 현시한다.
enable	쉘내부지령들을 가능 또는 불가능으로 한다.
eval [args]	인수들을 입력하기 위해 쉘로 읽어 들이고 그 결과에 생기는 지령을 실행한다.
exec 지령	이 쉘에서 지령을 실행한다.
exit[n]	상태 n을 가지고 쉘에서 탈퇴한다.
export[var]	var가 자식쉘에 알려 지게 한다.
fc	리력지령의 편집을 위한 history의 고정지령이다.
fg	배경일감을 전경에로 넘긴다.
getopts	지령행추가선택들을 구문해석하고 처리한다.
hash	지령을 빨리 찾기 위해 내부하쉬표를 조종한다.
help[지령]	내부지령에 대하여 도움이 되는 정보를 현시하고 지령이 지정되면 그 내부지령에 대한 구체적인 도움말을 현시한다.
history	행번호와 함께 history목록을 현시한다.
jobs	배경에 있는 일감들을 현시한다.
kill[-signal process]	신호를 프로쎄스의 PID번호나 일감번호에로 보낸다. 신호들의 목록을 위해 Kill-l을 입력한다.
let	연산식을 평가하고 산수연산의 결과를 변수에 대입하는데 리용한다.
Local	함수에 대한 변수들의 유효범위를 제한하기 위해 함수에서 리용된다.
logout	등록가입쉘에서 탈퇴한다.
popd	등록부탄창의 입력을 삭제한다.
pushd	등록부탄창에 입력을 추가한다.
pwd	현재작업등록부를 인쇄한다.
read[var]	표준입력으로부터 var변수에로 행을 읽어 들인다.
readonly[var]	변수 var를 읽기전용으로 만든다. 재설정할수 없다.

(표계속)

지 령	기 능
return[n]	되돌이값으로 주어 진 탈퇴값이 n 인 함수로부터 복귀된다.
set	추가선택들과 위치파라메터들을 설정한다(666페지에 있는 표 12-2를 참고).
shift[n]	위치파라메터들을 n번 왼쪽으로 자리밀기한다.
stop pid	프로쎄스번호 PID의 실행을 중지시킨다.
Suspend	현재쉘의 실행을 중지시킨다(그러나 등록가입쉘에서는 중지시키지 않는다.).
test	파일형태를 검사하고 조건식을 평가한다.
Times	이 쉘로부터 실행된 프로쎄스들을 중첩된 사용자와 체계시간을 인쇄한다.
trap[arg][n]	쉘이 신호 n(0, 1, 2 또는 15)을 받으면 arg를 실행한다.
type[command]	지령형태를 인쇄한다. 실례로 pwd는 내부쉘지령이다.
typeset	declare와 같다. 변수들을 설정하고 그것들에 속성을 준다.
ulimit	프로쎄스자원한계를 현시하고 설정한다.
umask[octal digits]	소유자, 그룹과 기타를 위한 사용자파일작성방식마스크를 설정한다.
unalias	별명들을 해제한다.
unset	[name]변수나 함수의 값을 해제한다.
wait[pid#n]	PID번호 n을 가진 배경프로쎄스를 대기하고 끝상태를 알려 준다.

Bash쉘련습

련습문제 49: 첫번째 스크립트

1. 다음의 기능을 수행하는 greetme스크립트를 작성하시오.

   ㄱ. 사용자의 이름, 스크립트의 이름, 사명을 설명문으로 포함한다.
   ㄴ. 사용자에게 인사한다.
   ㄷ. 날자와 시간을 인쇄한다.
   ㄹ. 이 달의 력서를 인쇄한다.
   ㅁ. 자신의 콤퓨터이름을 인쇄한다.
   ㅂ. 이름과 조작체계의 판본을 인쇄한다
   ㅅ. 어미등록부의 모든 파일목록을 인쇄한다.
   ㅇ. Root가 실행하는 모든 프로쎄스들을 인쇄한다.
   ㅈ. TERM, PATH, HOME변수값들을 인쇄한다.
   ㅊ. 디스크사용법을 인쇄한다.
   ㅋ. id지령을 사용하여 사용자의 그룹 ID를 인쇄한다.

ㅌ. please couldn’t you loan me $50.00?를 인쇄한다.
ㅍ. Good bye를 표시하고 현재시간을 표시한다.

2. 스크립트를 실행형으로 만드시오.

    Chmod +X greetme

3. 사용자의 스크립트의 첫번째 행은 무엇이였는가? 이 행이 왜 필요한가?

련습문제 50: 지령행인수

1. 2개의 인수를 가지는 rename스크립트를 작성하시오. 첫번째 인수는 초기파일의 인수이고 두번째 인수는 파일의 새로운 이름이다. 만일 사용자가 2개의 인수를 주지 않으면 사용법통보문이 현시장치에 나타나고 스크립트에서 탈퇴한다. 다음의 실례는 스크립트의 동작방식을 설명한다. 스크립트에서 탈퇴한다. 스크립트결과를 실례로 보여 주었다.

```
$ rename
Usage:rename oldfilename newfilename
$
$ rename file1 file2
file1 has been renamed file2
Here is a listing of the directory
A file2
File bak
```

2. 다음의 find지령(SunOS)은 100k보다 큰 root구획에 있는 파일들과 지난주에 수정된 파일들을 표시한다(체계에서 정확한 find문법을 위해 사용지도서페지들을 검사하시오.).

```
Find /*dev –mtime –7 –size +200 –print
```

3. 2개의 인수를 가지는 bigfiles스크립트를 작성하시오. 한개 인수는 mtime이고 다른 하나는 size값이다. 사용자가 2개 인수를 주지 않으면 적당한 오유통보문이 표준오유에 전송된다.
4. 시간이 있으면 vi를 위한 여유파일을 작성하는 vib스크립트를 작성하시오. 여유파일의 이름뒤에는 확장자 .bak가 추가된다.

련습문제 51: 사용자입력얻기

1. 다음의 기능을 수행하는 mosy스크립트를 작성하시오.

ㄱ. 사용자의 완전이름을 묻는다. 첫번째, 마지막, 중간이름을 묻는다.
ㄴ. 그의 이름을 가지고 사용자에게 인사한다.
ㄷ. 사용자의 생일을 묻고 그의 나이를 계산한다(expr리용).

ㄹ. 사용자의 등록가입이름을 묻고 그의 사용자 ID를 현시한다 (/etc/passwd로부터).

ㅁ. 사용자에게 home등록부를 알려 준다.

ㅂ. 사용자에게 현재 실행하고 있는 프로쎄스를 보여 준다.

ㅅ. 사용자에게 요일과 현재시간을 알려 준다. 결과는 다음과 같은 형식을 취해야 한다.

Today of the week is Tuesday and the Curente time is 04:07:38pm

2. datafile이라고 하는 본문파일을 작성한다(이 파일이 이미 제공되여 있지 않으면). 매 입력은 옹근두점으로 분리되는 마당들로 이루어 진다. 마당들은 다음과 같다.

ㄱ. 성과 이름
ㄴ. 전화번호
ㄷ. 주소
ㄹ. 생일
ㅁ. 로임

3. 다음의 기능을 수행하는 스크립트 lookup을 작성하시오.

ㄱ. 스크립트이름, 사용자이름, 날자, 스크립트작성의 리유 등을 설명문으로 포함한다. 이 스크립트를 작성하는 목적은 분류된 순서로 datafile을 현시하는것이다.
ㄴ. 성을 가지고 datafile을 분류한다.
ㄷ. 사용자에게 datafile의 내용을 보여 준다.
ㄹ. 사용자에게 파일안의 입력개수를 알려 준다.
ㅁ. 스크립트를 오유수정하기 위해 –x와 –v추가선택을 리용한다. 이 지령들을 어떻게 리용했는가? 그것들은 어떻게 다른가?

련습문제 52: 조건명령문

1. 다음의 기능을 수행하는 스크립트 checking을 작성하시오.

ㄱ. 지령행인수는 사용자등록가입이름을 가진다.
ㄴ. 지령행인수가 보장되였는가를 검사한다.
ㄷ. /etc/passwd파일안에 사용자가 있는가를 검사하고 있으면

Found <user> in the /etc/passwd file

없으면

No such user on your system

을 인쇄한다.

2. lookup스크립트에서 사용자에게 datafile에 입력을 추가하겠는가를 물어 보고 대답이 yes 혹은 no이면

ㄱ. 새로운 이름, 전화번호, 주소, 생일, 로임을 물어 본다. 매 항목은 다른 변수에 기억된다. 마당들사이에 :을 주어 정보를 datafile에 추가한다.

ㄴ. 성을 가지고 파일을 분류한다. 사용자에게 입력을 추가하였다는것을 알려 주고 행번호가 앞에 붙은 행을 보여 준다.

련습문제 53: 조건부와 파일시험

1. Checking을 다시 작성한다. 이름 지어 진 사용자가 /etc/passwd파일에 있는가를 검사한후 프로그람은 사용자가 등록가입되였는가를 검사한다. 가입된 경우 프로그람은 현재 기동중인 모든 프로쩨스를 현시하고 그렇지 않으면 사용자에게

   <user> is not logged on

   을 알려 준다.

2. let지령을 사용하여 성적모임을 평가한다. 스크립트는 사용자에게 그의 시험점수를 물어 본다(declare –i를 리용하시오.). 스크립트는 성적값이 허용범위 0~100사이에 있는가를 검사한다. 만일 그사이에 놓여 있지 않으면 프로그람은 탈퇴한다. 만일 성적이 범위안에 있다면 사용자물음표성적이 현시된다. 실례로 You receive an A, Excellent!

   범위는 다음과 같다.

   A(90-100), B(80-89), C(70-79), D(60-69), F(60이하)

3. lookup스크립트는 datafile에 의존하여 실행된다. 여기서 datafile이 있는가 그리고 읽고쓰기할수 있는가 검사하시오. lookup스크립트에 다음과 같은 차림표를 추가하시오.

   [1] 입력추가
   [2] 입력지우기
   [3] 입력보기
   [4] 탈퇴

   스크립트에는 이미 입력추가부분이 있다. 입력추가프로그람은 datafile에 이름이 이미 있는가를 검사하는 코드를 포함해야 한다. 이름이 없으면 새로운 입력을 추가한다. 입력지우기, 입력보기 그리고 탈퇴기능에 대한 코드를 작성하시오. 스크립트의 지우기부분은 먼저 그것을 지우기전에 입력이 존재하는가를 검사한다. 존재하면 사용자에게 통보한다. 그렇지 않으면 입력을 지우고 사용자에게 그것을 지웠다는것을 알려 준다. 탈퇴할 때 한개 수자를 리용하여 적당한 탈퇴상태를 표시하시오.

련습문제 54: Case명령문

1. PS지령은 BSD(Berkeley UNIX)와 System 5(AT&T UNIX)에서 서로 다르다. UNIX는 ps에 대하여 BSD추가선택을 리용한다. System 5에서 모든

프로쎄스를 현시하는 지령은 ps-ef이다. UNIX에서 지령은 ps aux이다. 서로 다른 체계형의 번호를 검사하는 systype프로그람을 작성하시오. 검사항목은 다음과 같다.

AIX
UNIX
HP-UX
SCO
OSF1
ULTRIX
SunOS(solaris / SunOs)
OS

Solsris, HP-UX, SCO, IRIX는 AT&T형체계이다. 나머지는 BSD형이다. 지금 리용하는 UNIX의 판본이 stdout로 인쇄된다. 체계이름은 uname-s지령 또는 /etc/motd파일로부터 알아 낼수 있다.

2. 다음의 기능을 수행하는 timegreet스크립트를 작성하시오.

ㄱ. 스크립트우에 이름, 날자, 목적을 설명문으로 첨가하시오.
ㄴ. 다음의 프로그람을 변화시켜 if/elif대신에 case지령을 리용하시오.

```
#! /bin/bash
# comment section
  you=$LOGNAME
  hour=$(date+%H)
echo "The time is:$(date+%T)
if ((hour>0 && hour<12))
 then
    echo "Good Morning $you!"
  elif ((hour==12))
  then
    echo "Lunch time!"
  elif ((hour>12 && hour<16))
  then
    echo "Good afternoon, $you!"
  else
    echo "Good night, $you, Sweep dreams."
```

련습문제 55: 순환

다음의것들중 하나를 선택하시오.

1. 새로운 우편을 검사하고 그것이 도착했으면 현시장치에 통보를 보내는 mchecker라고 하는 프로그람을 작성하시오.

ㄱ. 프로그람은 사용자를 위해 우편스플파일의 크기를 얻는다(스플파일은 AT&T체계에서는 /usr/mail/$LOGNAME에 있고 UNIX와 USB체계에서는 /usr/spool/mail/$USR에 있다.). 스크립트는 30s마다 순환을 계속 진행한다. 매번 순환이 진행될 때마다 그것은 우편스플파일의 크기를 이전 순환에서의 크기와 비교한다. 만일 새로운 크기가 이전의 크기보다 더 크면 통보 Username, you have new mail.을 현시장치에 내보낸다. 파일의 크기는 ls-l, wc-c지령이나 find지령의 출력을 보고 알수 있다.

2. 다음과 같은 기능을 수행하는 스크립트를 작성하시오.

ㄱ. 사용자이름, 날자, 프로그람의 목적을 스크립트우에 있는 설명문에 넣으시오.
ㄴ. 음식차림표를 만들기 위해 select순환을 사용하시오.
ㄷ. 출력은 다음과 같이 한다.

1) 불고기와 감자
2) 물고기와 감자튀기
3) 국과 쌀라드
선택하시오.1
푸짐한 식사
콜레스테롤에 주의를 돌리시오.
많이 드시오.

1) 불고기와 감자
2) 물고기와 감자튀기
3) 국과 쌀라드
선택하시오.2
영국료리
많이 드시오.

1) 불고기와 감자
2) 물고기와 감자튀기
3) 국과 쌀라드
선택하시오.3
건강식품들
치료식사는 싫증난다.
많이 드시오.

3. 사용자들의 목록에 한번에 하나씩 그것들이 현재 리용하고 있는 블로크수목록을 보내는 dusage를 작성하시오. 사용자목록은 potential_hogs파일에 있다. Potential_hogs파일에 있는 사용자들중 한명은 admin이다.

ㄱ. potential_hogs파일이 존재하는가, 읽기가능한가를 검사하기 위해 파일시험을 리용하시오.
ㄴ. 순환을 리용하여 사용자들의 목록을 반복한다. 500개 블로크이상을 리용하는 사용자들에게만 우편을 보낸다. 사용자 Admin는 빠진다(즉 그

는 우편통보를 받지 못한다.). 우편통보는 dusage스크립트의 here 문서에 있다.

ㄷ. 우편을 받은 사람의 이름목록을 작성한다. log파일을 만들어 이것을 수행하시오. 매 사람에게 우편을 보낸후 우편을 받은 사람의 수와 그 이름목록을 인쇄하시오.

련습문제 56: 함수

1. systype프로그람을 체계이름을 되돌리는 함수로 다시 작성하시오. 이 함수를 리용하여 checking프로그람에서 PS지령과 함께 어떤 추가선택들을 리용하겠는가를 결정하시오.
2. AT&T UNIX에서 모든 프로쎄스를 표시하는 ps지령은 ps-ef이다.
3. UNIX/BSD UNIX에서 지령은 ps -aux 혹은 ps aux[9]이다.
4. 모든 림시파일을 지우고 스크립트를 탈퇴하는 cleanup함수를 작성하시오. 만일 새치기신호나 정지신호가 프로그람실행중에 전송되면 trap지령은 cleanup함수를 호출한다.
5. here문서를 리용하여 lookup스크립트에 다음과 같은 새로운 차림표항목을 추가하시오.

   [1] 입력추가
   [2] 입력지우기
   [3] 입력변경
   [4] 입력보기
   [5] 입력탈퇴

   차림표에서 매 항목을 처리하는 항목을 작성하시오. 사용자가 유효입력을 선택하고 함수를 완성하였을 때 차림표를 다시 보려고 하는가를 물어 보시오. 만일 무효입력이 들어 있다면 프로그람은 다음의것을 인쇄하고 차림표가 다시 현시된다.

   Invalid entry, try again
6. 입력보기밑에 부분차림표를 작성하시오. 사용자에게 선택된 개별적인물들에 대한 특수정보를 보려고 하는가를 물어 보시오.

   ㄱ.전화번호
   ㄴ.주소
   ㄷ.생년월일
   ㄹ.로임
7. 스크립트에서 trap지령을 리용하여 프로그람실행도중에 새치기신호가 전송되면 지우기동작을 진행하시오.

---

9. UNIX에서 안내횡선을 쓰면 경고문이 발생한다(사용지도서페지를 참고).

제 13 장. 대화형TC쉘

제 1 절. 서론

대화형쉘은 표준입력, 표준출력 그리고 표준오유들이 하나의 말단에 련결되는 쉘이다. TC쉘 (tcsh) [1]을 대화형으로 리용할 때 tcsh재촉문에서 지령들을 입력하고 응답을 기다린다. TC쉘은 등록가입할 때 기동하여 지령들을 해석하는 프로그람이다. 이것은 이전의 판본인 Berkeley UNIX쉘을 개선한 공개령역프로그람이다. 추가된 특성은 지령행편집, 특색 있는 재촉문들, 프로그람적완성(파일이름, 지령 그리고 변수들), 맞춤법 등이다.

초기의 tcsh원천은 ftp.astron.com이나 ftp.gw.com 과 ftp.private.edu[2]에서 배포되였다. 비록 tcsh가 대부분 Linux의 배포물에 포함되여 있지만 Solaris나 Windows NT, HP-UX, QNX 등을 비롯한 많은 조작체계들과도 결합될수 있다.

이 장에서는 TC쉘을 대화형으로 리용하는 방법과 사용자초기작업환경을 설정하는 방법에 대하여 중점을 두고 설명한다. Berkeley C쉘에 비하여 개선된 특성은 이 장에서 알게 되는바와 같이 쉘과 대화형으로 작업할 때 보다 뚜렷이 나타난다. 그렇지만 프로그람작성언어로서 리용될 때에는 tcsh와 csh가 사실상 꼭같다. 다르다면 /bin/csh 대신에 /bin/tcsh를 리용하는것이다.

1. tcsh의 판본

리용하고 있는 tcsh의 판본을 알기 위하여 쉘재촉문에서 다음과 같이 입력한다.

Which tcsh

tcsh가 어느 등록부에 설치되였는지 알기 위하여(보통은 /bin) 그리고 판본정보를 인쇄하기 위하여 다음과 같이 입력한다.

/directory_path/tcsh_c'echo $version'

실례 13-1

1. **which tcsh**
   /bin/tcsh
2. **/bin/tcsh -c 'echo $version'**
   tcsh 6.09.09 {Astron} 1998-08-16 (sparc-sun-solaris) options 8b,nls,dl,al,rh,color

---

[1] tcsh에서 T는 PDP-10콤퓨터를 위해서 DEC가 사용한 TENEX와 TOP-10의 조작체계에서 유래되였다. 이 체계들은 감시기에 대한 지령완성의 형식을 가지고 있다. Tcsh의 작성자는 이 체계들의 특징들을 계승하여 C쉘에 T를 추가하였다.

[2] www.tac.nyc.ny./mirrors/tcsh-book에서 자습용참고서를 보아 주기를 바란다..

2. 기동

TC쉘이 재촉문을 현시하기전에 몇가지 프로세스가 먼저 진행된다. 이 과정을 그림 13-1에 주었다.

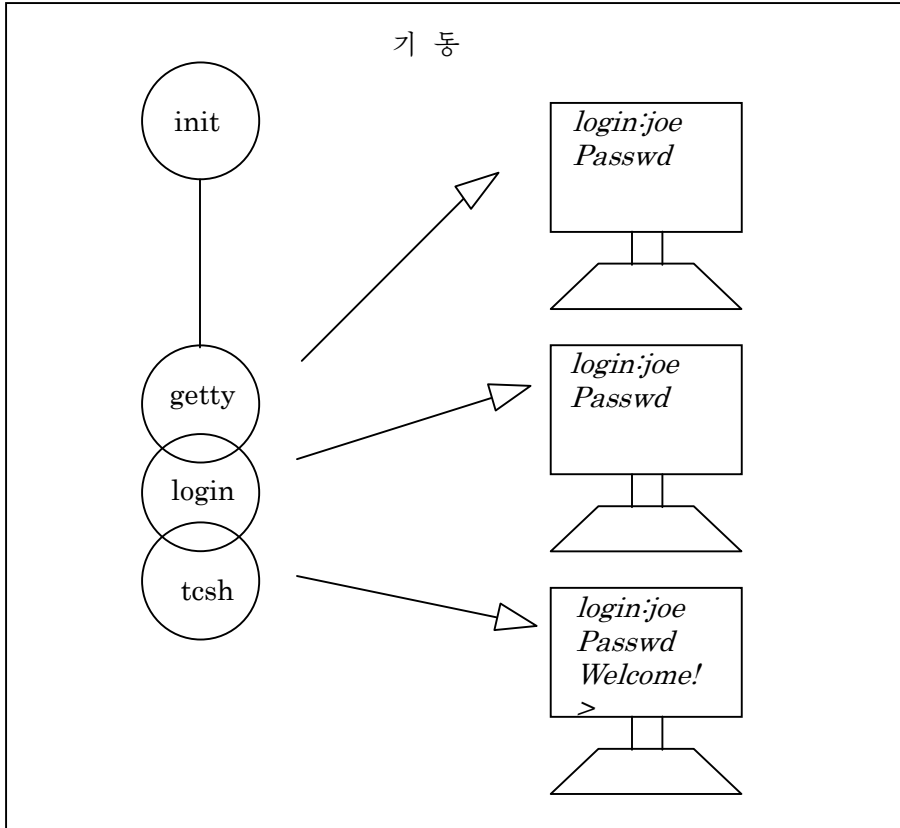



그림 13-1. 체계기동과 TC 쉘

체계가 기동한후 실행되는 최초의 프로쎄스를 초기화(init) 즉 프로쎄스식별번호(PID)라고 부른다. 그다음 getty프로쎄스가 실행된다. 이 프로쎄스들은 말단포구들을 열어 입력이 들어 오는 곳과 표준출력과 오유가 보낼 곳을 제공하며 현시장치에 login재촉문을 내보낸다. 사용자가 자기의 이름을 입력한후에 /bin/login프로그람이 실행된다. login프로그람은 암호를 물어 보고 그것을 검증하고 초기작업환경을 설정한 다음에 쉘 /bin/tcsh를 초기화한다. TC쉘은 /etc/csh.cshrc와 /etc/csh .login(만일 있다면)라고 불리우는 체계기동파일을 /etc등록부에서 찾아 본다. 다음에 쉘은 사용자의 홈등록부에서 tcsh환경을 전용화하는데 리용되는 또 다른 초기파일인 ~/.tcshrc 파일을 찾는다. 그 파일이 없으면 ~/.cshrc라고 부르는 같은 일감을 수행하는 다른 파일(csh를 실행할 때 보통 요구된다.)을 찾는다. .tcshrc파일안에 있는 지령들을 실행한후에 일반적으로 .history라고 불리우는 리력파일을 실행한다. 그다음 ~/.login파일안에 있는 지령들이 실행되고 끝으로 .cshdirs파일이 실행된다. 이 매개 파일들은 750페지의 《TC쉘환경》

에서 설명하였다.[3] /etc/csh.cshrc와 ~/.tcshrc파일들은 새로운 TC쉘이 시작될 때마다 실행된다. .login파일은 오직 사용자가 등록가입할 때에만 실행되는데 이 파일에는 사용자의 환경을 초기화하기 위한 지령과 변수들을 포함한다. 모든 기동파일들로부터 지령들이 실행된후에 재촉문(표준으로 > 이다.)이 현시장치에 나타나며 tcsh는 지령을 기다린다. 그림 13-2에서 보여 준다.

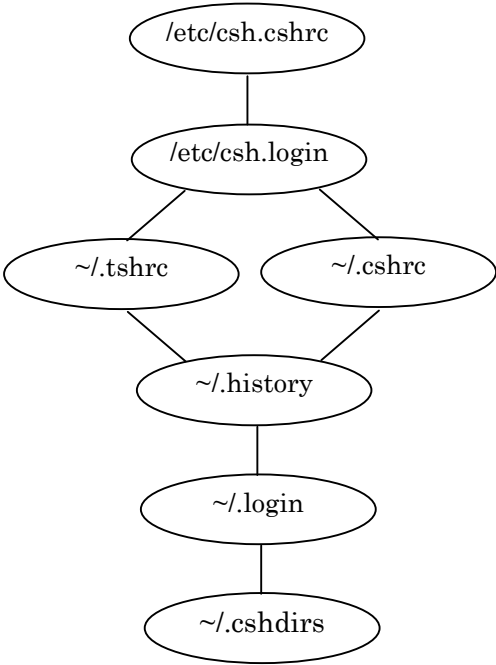



그림 13-2. 초기화파일들의 원천화순서

등록탈퇴할 때 사용자는 Control-D를 누르거나 autologout쉘변수가 설정되여 있다면 자동적으로 등록탈퇴된다. 등록탈퇴하기전에 쉘은 홈등록부안에서 /etc/csh.logout 또는 ~/.logout라고 불리우는 파일을 찾으며 있으면 그의 지령들이 실행된다.

.logout파일에는 보통 림시파일을 지우고 log파일에 자료를 추가하거나 사용자에게 인사를 하는 등의 지령들이 있다.

제 2 절. TC쉘환경

1. 초기화파일

tcsh프로그람이 실행된후에 체계내의 기동파일, /etc/csh.cshrc를 실행하고 사용자의 홈등록부에 있는 쉘초기화파일들인 .tcshrc파일을, 그다음에 login파일을 실행하도록 프로그람이 작성된다.

이 파일들은 사용자가 자기의 자체환경을 초기화하게 한다.

---

[3] 이 파일들을 읽는 순서는 tcsh가 번역될 때 변화될수 있다.

실례 13-2

```
# /etc/csh.cshrc
# Systemwide environment and startup programs for csh users
1.  if ($?PATH) then
2.  setenv PATH "${PATH}:/usr/X11R6/bin"
     else
3.  setenv PATH "/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin" endif
4.  if ($?prompt) then
5.          [ "$SHELL" = /bin/tcsh ]
6.          if ($status == 0) then
7.                  set prompt='[%n@%m %c]$ '
8.          else
9.                  set prompt=\ ['id -nu' @' hostnalfte -s'\ ]\ $\
10.         endif
            endif
11.   limit coredumpsize 1000000

12.  [ 'id -gn' = 'id -un' -a 'id -u' -gt 14 ]
13.  if $status then
14.  umask 022
      else
15.  umask 002
      endif
16.  setenv HOSTNAME '/bin/hostname'
17.  set history=1000

18.  test -d /etc/profile.d
19.      if ($status == 0) then
20.          set nonomatch
21.                foreach i ( /etc/profile.d/*.csh )
22.                    test -f $i
                  if ($status == 0) then
23.                                 source $i
       endif
       end
24.  unset nonomatch
      endif
```

설명

1. $?PATH는 PATH변수가 설정되였는가를 보기 위한 검사항이다. 설정되였으면 1을 되돌린다.
2. PATH변수가 설정되였다면 /usr/x11R6/bin은 그 변수에 추가된다. 이것은

X windows파일을 포함하는 등록부이다.

3. PATH변수가 이전에 설정되지 않았다면 이 행은 그 변수를 /bin:/usr/bin:/usr/x11R6/bin으로 설정한다.
4. 이 행은 재촉문이 설정되였는가를 검사한다.
5. 중괄호안에 식을 써넣으면 그 식이 검사되며 식의 결과가 참이면 령탈퇴상태로 귀환되고 그렇지 않으면 령이 아닌 탈퇴상태로 귀환한다. SHELL환경변수의 값이 /bin/tcsh이면 탈퇴상태는 령이다.
6. status변수가 실행되는 마지막지령의 탈퇴상태를 포함하는데 이 실례에서는 5행의 이전 test지령이다.
7. 마지막지령으로부터의 상태가 령이였다면 /bin/tcsh를 위한 재촉문이 설정된다. 그 재촉문은 @기호, host이름, 현재작업등록부(모든것이 []안에 있다.) 그리고 화폐기호가 뒤에 놓이는 사용자이름으로 설정된다.
8. 상태가 령이 아니였다면 else는 9행으로 프로그람조종을 넘긴다.
9. 이 행은 표준csh프로그람을 위한 재촉문을 설정한다. 이것은 사용자이름(id-un), @기호, 사용자의 주콤퓨터의 짧은 이름 즉 첫번째 점에서 잘린 호스트이름과 $을 인쇄한다.
10. endif는 if블로크를 완료한다.
11. ore파일의 크기(보통 프로그람이 일부 비법적인 체계조작때문에 파괴될 때 만들어 지는 파일)는 1,000,000byte로 제한된다. core파일은 실행중의 프로그람을 Control-\ 로 끝낼 때 만들어 진다.
12. 그룹 ID번호와 사용자ID번호가 같으면 그리고 사용자 ID번호가 14보다 더 크면 다음행이 실행되고 그렇지 않으면 else아래행이 실행된다. 대표적으로 14이하의 사용자 ID는 root, daemon, and, lp 등과 같은 특수한 사용자를 위한것이다(/etc/passwd에서 보여 준다. 사용자 ID는 마당번호 3에 있다.).
13. 이전 행에서의 검사가 령이 아닌 탈퇴상태를 되돌리면 14행이 실행된다. 그렇지 않으면 15행이 실행된다.
14. umask지령은 파일작성마스크를 설정한다. 즉 작성될 때 파일들과 등록부를 위한 초기허가를 설정한다. 작성될 때 등록부는 755(rwxr-xr-x),파일은 644(rw-r-r—)를 얻는다.
15. umask는 등록부가 만들어 질 때 그의 허가를 755로 되게, 파일은 644(rw-rw-r-)를 얻도록 설정된다.
16. 환경변수 HOSTNAME에 /bin/hostname지령의 출력이 대입된다.
17. history변수는 1000으로 설정된다. 지령이 지령행에 입력될 때 그것들은 리력목록에 기억된다. history변수를 1,000으로 설정하면 1,000개이상의 지령들은 history지령이 입력될 때 표시되지 않는다.
18. test지령은 /etc/profile.d등록부가 존재하면 0상태, 존재하지 않으면 0이 아닌 탈퇴상태를 되돌린다.
19. 상태가 0이면 등록부는 존재하고 프로그람은 20행으로 이행한다.
20. nonomatch변수는 어떤 특수한 메타문자(*, ?, [])가 대조할수 없다면 쉘이 오유통보를 보내지 않도록 설정된다.
21. foreach순환은 /etc/profile.d등록부안에 있는 .csh확장자를 가진 매 파일이 검사될 때까지 차례로 변수 i에 대입한다(22행-24행).
22. 변수 i에 대입된 파일이름이 정규파일(-f)이라면 다음행으로 간다.

23. test가 0상태(참)로 귀환되였다면 그 파일을 원천화한다. 즉 현재환경에서 그것을 실행한다.
24. End예약어는 순환본체의 끝을 의미한다.

~/.tcshrc파일 tcshrc파일은 tcsh변수설정을 포함하며 tcsh자식쉘이 시작될 때마다 실행된다. 별명과 리력은 보통 여기에서 설정된다.

실례 13-3

```
(The .tcshrc File)
1.  if ( $?prompt ) then
2.  set prompt = "\ ! stardust >
3.  set history = 100
4.  set savehist = 5
5.  set noclobber
6.  set rmstar

7.  set cdpath. =( /horne/3ody/ellie/bin /usr/local/bin /usr/bin )
8.  set ignoreeof
9.  alias m more
     alias status 'date;du -s'
     alias cd 'cd \ !*;set proinpt = "\ i <$cwd>"'
10.  endif
```

설명

1. 재촉문이 설정되여 있다면($?prompt) 쉘은 대화형으로 실행된다. 즉 script 안에서 실행되지 않는다. 재촉문은 대화형쉘에 대해서만 설정된다.
2. 1차재촉문은 현재리력사건의 수, 이름 stardust 그리고 >문자로 설정된다. 이것은 기정값인 >재촉문을 변경시킨다.
3. history변수는 100으로 설정된다. 이것은 현시장치에 나타날 리력사건의 수자를 조종한다. 마지막 100개의 지령들은 history를 입력할 때 현시된다(763페지의 《리력》에서 보여 준다.).
3. 일반적으로 등록탈퇴할 때 리력목록은 지워 진다. savehist변수는 리력목록의 끝으로부터 지정된 개수의 지령들을 기억할수 있게 한다. 이 실례에서 마지막 5개 지령이 홈등록부에 있는 파일 즉 history파일에 기억되여 다시 등록가입할 때 쉘은 그 파일이 존재하는가를 검사하고 기억된 리력행들을 새로운 리력목록의 꼭대기에 넣을수 있다.
5. noclobber변수는 방향바꾸기를 리용할 때 사용자가 부주의로 파일을 지우지 않도록 보호한다. 실례로 sort myfile>myfile은 myfile을 파괴한다. noclobber를 설정했을 때 출력을 존재하는 파일에 방향바꾸기하려고 하면 파일이 존재한다는 통보문이 나타난다.
6. Tcsh변수 rmstar가 설정되면 사용자에게 실지로 rm*을 입력한후에 모든 파일을 지우려고 하는가를 물어 본다. 즉 사용자는 현재작업등록부에 있는 모든 파일들을 지우는것을 막을수 있다.

7. cdpath변수에는 경로요소들의 목록이 대입된다. 등록부를 변경시킬 때 만일 방금 등록부이름을 지정하고 그 등록부가 현재작업등록부아래에 놓인 보조 등록부가 아니라면 쉘은 cdpath등록부입력을 검사하여 그 위치에서 등록부를 찾을수 있는가를 보고 다음에 등록부를 변화시킨다.
8. ignoreof변수는 ^D로 등록탈퇴하지 못하게 한다. mail프로그람과 같은 건반으로부터 입력을 접수하는 UNIX편의프로그람들은 ^D를 입력하면 완료된다. 낮은 체계에서는 흔히 사용자가 한번이상 ^D를 입력하게 된다. 첫번째에는 mail프로그람이 완료되지만 두번째는 사용자가 등록탈퇴된다. ignoreof를 설정하면 logout 를 입력하여 등록탈퇴할수 있다.
9. aliases는 단일 지령이나 지령그루빠를 위해 속기표시를 할수 있도록 설정된다. 별명을 입력할 때 그에 대입된 지령들이 실행된다. more지령에 대한 별명은 m이다. m을 입력할 때마다 more지령이 실행된다. status별명은 날자와 사용자디스크사용법을 인쇄한다. cd별명은 사용자가 등록부를 변경할 때마다 새로운 재촉문들을 작성한다. 새로운 재촉문은 현재리력사건(\ !*)의 수와 < >안에 있는 현재작업등록부($cwd)를 포함한다.
10. endif는 1행에서 시작된 if명령블로크의 끝을 표시한다.

~/.login파일 login파일은 처음 등록가입할 때 한번 실행된다. 이 파일에는 일반적으로 환경변수와 말단설정이 들어 있다. 이 파일은 window응용프로그람들이 보통 시작되는 파일이다. 환경변수들은 이 쉘에서 파생된 프로쎄스들에 의해 계승되여 한번만 설정할 필요가 있으며 말단설정이 매 프로쎄스들에 의해 재설정되지 않기때문에 이 설정들은 .login파일에 속한다.

실례 13-4

```
(The .login File)
1.  stty -istrip
2.  stty erase ^h
3.  stty kill ^u
    #
    # If possible start the windows system.
    # Give a user a chance to bail out
    #
4.  if ( $TERM == "linux" ) then
5.          echo "Starting X windows.   Press control C \
                 to exit within the next 5 seconds "
           sleep 5
6.          startx
7.  endif
8.  set autologout=60
```

설명

1. stty지령은 추가선택들을 설정한다. 입력문자들은 만일 -istrip가 리용된다면 7개 비트로 끊어 지지 않는다.

2. Stty지령은 Backspace건인 Control-H를 설정한다.
3. #로 시작하는 임의의 행은 설명문이다. 그것은 실행명령이 아니다.
4. 현재말단창문(tty)이 조종탁(linux)이면 다음행이 실행되며 그렇지 않으면 프로그람조종은 마지막 endif로 간다.
5. 이 행은 현시장치에 출력되며 사용자가 프로쎄스를 끝내기 위하여 Control-C건을 누르지 않는다면 프로그람은 5s동안 정지되며 그때 Xwindows프로그람을 시작한다.
6. startx프로그람은 X windows를 시작한다.
7. endif는 제일 안쪽 if구조의 끝을 표시한다.
8. autologout변수는 60으로 설정되여 60분이 지나서 사용자가 자동적으로(등록가입쉘로부터) 등록탈퇴된다.

2. 탐색경로

path변수는 쉘에 의해 리용되여 지령행에 입력된 지령들을 찾는다. 탐색은 왼쪽에서 오른쪽으로 진행된다. 점(.)은 현재작업등록부를 나타낸다. 만일 지령이 경로에 표시된 등록부들이나 현재작업등록부에서 찾을수 없으면 쉘은 Command not found통보문을 표준오유에 보낸다. 경로는 보통 .login파일에[4] 설정된다. TC쉘에서 탐색경로는 Bash와 Korn쉘에서와 다르게 설정된다. 매 요소는 TC쉘에서 공백에 의해 분리되지만 다른 쉘들에서는 웅근두점에 의해 구분된다.

TC쉘은 PATH가 이 쉘로부터 파생되고 path변수를 리용할 필요가 없는 Bash, Bourne, Korn쉘들과 같은 다른 프로그람들과 호환성을 유지하도록 환경변수를 내부적으로 갱신한다.

실례 13-5

```
# Path is set in the ~/.kcshrc file.
1.  set path = (/usr/bin /bin /usr/bsd /usr/local/bin    .)
2.  echo $path
    /usr/bin /bin /usr/bsd /usr/local/bin .
    # The environment variable PATH will display as a colon-    separated list
3.  echo $PATH
    /usr/bin:/bin:/usr/bsd:/usr/1ocal/bin:.
```

설명

1. 탐색경로는 tcsh를 위해 설정된다. 그것은 어떤 지령이 지령행에서 입력될 때 tcsh에 의해 왼쪽에서 오른쪽으로 탐색되는 등록부들의 공백으로 구분된 목록으로 이루어 져 있다. 탐색경로는 현재쉘에 국한된다.
2. path변수의 값이 현시된다.

---

[4] 탐색경로변수를 .cshrc 파일에서 설정된 cdpath 변수와 혼돈하지 마시오.

3. 환경변수 PATH의 값이 표시된다. 그것은 path변수에 렬거된것과 같은 등록부의 두점으로 구분된 목록인데 현재쉘로부터 호출된 다른 프로그람이나 응용프로그람에로 보낸다(bash, sh, ksh는 경로를 옹근두점에 의해 분리되는 목록으로 설정한다.).

rehash지령 쉘은 탐색경로에 표시된 등록부의 내용들로 이루어 진 내부하쉬표를 만든다(만일 점이 탐색경로안에 있다면 현재작업등록부인 점등록부에 있는 파일은 하쉬표에 넣어 지지 않는다.). 효률을 높이기 위해 쉘은 매번 경로를 탐색하지 않고 지령행에 입력된 지령을 하쉬표를 리용하여 찾는다. 새로운 지령이 탐색경로에 이미 표시된 등록부중의 하나에 추가되면 내부하쉬표는 다시 계산되여야 한다. 이것을 다음과 같은 지령을 입력하여 수행한다.

rehash

하쉬표는 또한 경로를 변경하거나 다른 쉘을 시작할 때 자동적으로 다시 계산된다.

hashstat지령 이 지령은 하쉬표로부터 지령을 탐색하는 효과를 보여 주기 위하여 성능통계를 현시한다. 통계는 《성공》과 《실패》의 형식으로 되여 있다. 만약 쉘이 리용된 대다수의 지령들을 경로의 끝에서 찾으면 그것들이 경로의 앞에 있을 때보다 찾기가 더 힘들어 성공보다 실패를 더 많이 하게 된다. 이런 경우에 성능을 개선하기 위하여 경로의 앞쪽에 성공하기가 제일 힘든 등록부를 놓는다.[5] unhash내부지령은 내부하쉬표를 리용하지 못하게 한다.

> hashstat
1024 hash bucket of 16 bit each

source지령 이 지령은 내부쉘지령 즉 쉘의 내부코드부분이다.

그것은 파일로부터 지령이나 지령묶음을 실행하는데 리용된다. 보통 어떤 지령이 실행될 때 쉘은 자식프로쎄스를 파생시켜 그 지령을 실행하여 임의의 변화가 어미쉘이라고 하는 초기쉘에 영향을 미치지 않도록 한다. source지령은 프로그람이 현재의 쉘에서 실행되도록 함으로써 파일안에서 설정된 임의의 변수들이 현재쉘환경의 부분으로 되도록 한다. source지령은 보통 .tcshrc, .cshrc 또는 .login중의 어느 하나가 수정되면 그것을 재실행하기 위하여 리용된다. 실례로 path가 등록가입한후에 변경되였다면 다음과 같이 입력한다.

>source .login 또는 source .tcshrc

3. 쉘재촉문

TC쉘은 3개의 재촉문 즉 1차재촉문(>기호), 2차재촉문(while, foreach나 if와 같은 tcsh지령이 뒤에 오는 물음표[?]) 그리고 맞춤법에 리용되는 3차재촉문을 가지고 있다. 1차재촉문은 등록가입한후에 말단에 표시되는 재촉문이다. 그것을 재설정할수 있다. 만일

[5] vfork(2)가 없는 콤퓨터에서는 hash 바키트의 번호와 크기를 인쇄한다.

결심채택이나 순환과 같은 tcsh프로그람작성구조를 요구하는 재촉문에서 스크립트를 작성하고 있다면 2차재촉문이 나타나 다음행에서 계속 할수 있게 한다. 그것은 구조가 완료될 때까지 매 행바꾸기문자다음에 계속 나타난다. 3차재촉문은 맞춤법검사가 설정되였으면 현시장치에 나타나 자동맞춤법을 확인한다. 그것은 문자렬 CORRECT>correct command(y|n|e|a)?를 포함한다. 재촉문들은 재촉문문자렬에 특수형식화문자렬을 추가하여 전용화할수 있다. 표 13-1에 재촉문문자렬을 주었다.

표 13-1. 재촉문문자렬

문자렬	의 미
%/	현재작업등록부
%~	현재작업등록부. 여기서 ~는 사용자의 홈등록부를 표시하고 다른 사용자의 홈등록부는 ~사용자에 의해 표시된다.
%c[[0]n],%.[[0]n]	현재작업등록부의 마지막요소 또는 n(수자)이 주어 지면 n개의 마지막요소들
%C	%c와 같지만 ~바꿔넣기를 하지 않는다.
%h,%!,!	현재리력사건번호
%M	완전한 호스트이름
%m	첫번째 《.》까지의 호스트이름
%S(%s)	두드러진 방식을 시작(정지)
%B(%b)	굵기방식시작(정지)
%U	(%u)밑선방식시작(정지)
%t,%@	12시간AM/PM형식의 시간
%T	%t와 같지만 24시간형식이다.
%p	12시간 AM/PM형식에서 초까지 표시하는 정확한 시간
%P	%p와 같지만 24시간형식이다.
^c	c는 맺기건에서와 같이 구문해석된다.
\ c	c는 맺기건에서와 같이 구문해석된다.
%%	한개의 %이다.
%n	사용자이름
%d	Day형식의 요일
%D	dd형식의 날자
%w	Mon형식의 달

(표계속)

문자렬	의 미
%W	mm형식의 달
%W	mm형식의 달
%y	yy형식의 년도
%Y	yyyy형식의 년도
%l	쉘의 tty
%L	재촉문의 끝으로부터 표시끝이나 또는 행의 끝까지 지운다.
%$	$뒤에 있는 쉘이나 환경변수이름을 즉시에 확장한다.
%#	보통 사용자에 대해서는 >(또는 promptchars쉘변수의 첫번째 문자)이고 #(또는 promptchars의 두번째문자)는 주사용자에 대한것이다.
%[string%]	한개의 문자탈출문자로서 문자렬을 포함한다. 말단속성들을 변화시킬 때에만 사용되며 유표위치를 이동시키지 않는다.
%?	재촉문전에 실행되는 지령의 되돌이코드
%Rprompt	2에서 구문해석기의 상태이다. prompt3에서 정정된 문자렬이고 history에서 리력문자렬이다.

1차재촉문 대화형으로 실행할 때 재촉문은 지령을 입력하고 enter건을 누르기를 기다린다. 만약 기정재촉문을 리용하고 싶지 않다면 .tcshrc파일에서 재설정하는데 그것은 이 쉘과 그로부터 파생되는 모든 TC쉘에 대하여 설정된다. 만약 이 등록가입대화조종을 하기 위해서만 설정하려고 한다면 그것을 쉘재촉문에서 설정한다.

실례 13-6

```
1. > set prompt = ' [ %n@%m %C] #'
2. [ ellie@homebound ~]# cd ..
3. [ ellie@homebound /home]# cd ..
```

설명

1. 1차재촉문에 가입자이름(%m), 공백그리고 현재작업등록부가 뒤따르는 사용자의 등록가입이름(%n)을 대입한다. 문자렬은 #가 뒤따르는 중괄호로 둘러싸인다.
2. 새로운 재촉문이 표시된다. 재촉문에서 나타나는 ~은 사용자의 홈등록부를 의미한다. cd지령은 등록부를 어미등록부로 변경한다.
3. 새로운 재촉문은 현재작업등록부/home을 가리킨다. 이 방법으로 사용자는 항상 자기가 어느 등록부에 있는가를 안다.

2차재촉문 2차재촉문은 재촉문에서 직결스크립트를 작성할 때 나타난다. 2차재촉문은 변경시킬수 있다. 쉘프로그람작성지령들이 입력되고 그다음에 행바꾸기문자가 올 때마다 2차재촉문이 나타나 그 지령이 완전히 끝날 때까지 계속 나타난다. 재촉문에서 스크립트를

정확히 작성하려면 많이 작성해 보아야 한다. 일단 지령이 입력되고 Enter건을 누르면 여유를 만들수 없고 tcsh리력기구는 2차재촉문에서 입력된 지령들을 기억하지 않는다.

실례 13-7

1. > foreach pal (joe tom ann)
2. **forreach?** echo Hi $pal
3. **forreach?** End

   Hi joe

   Hi tom

   Hi ann
4. >

설명

1. 이것은 즉시스크립트를 작성하는 실례이다. TC 쉘이 foreach순환이 시작된 다음에 입력을 더 요구하기때문에 2차재촉문이 나타난다. foreach순환은 괄호안에 있는 목록의 매 단어를 처리한다.
2. 첫번째로 순환할 때 joe는 변수 pal에 대입되고 문자렬 hijoe가 표시된다. 두번째 순환때 tomol이 변수 pal에 대입되며 이런 식으로 순환이 계속된다.
3. end명령은 순환의 끝을 표시한다. 괄호안에 있는 목록의 모든 항목들이 처리되였을 때 순환이 끝나고 1차재촉문이 표시된다.
4. 1차재촉문이 표시된다.

실례 13-8

1. > **set propmpt2='%R %%'**
2. > **foreach name (joe tom ann)**
3. foreach % **echo Hi $name**
4. foreach % end

   Hi joe>

   Hi tom

   Hi ann
5. >

설명

1. 2차재촉문인 prompt2는 %R가 1차재촉문에서 행 2에 입력된 조건부이름이거나 순환구조의 이름으로 되는 형식화된 문자렬로 재설정된다. 2개의 퍼센트기호는 한개의 퍼센트기호와 같다.
2. foreach지령이 시작되였다. 이것은 예약어 end로 끝나야 하는 순환구조이다. 2차재촉문이 순환이 완전히 끝날 때까지 계속 나타난다.
3. 2차재촉문은 foreach%이다.

4. end예약어가 입력된 다음 순환이 실행된다.
5. 2차재촉문이 다시 나타나 사용자입력을 대기한다.

4. 지령행

등록가입한후 TC쉘은 기정으로 1차재촉문을 >기호로 표시한다. 쉘은 지령해석기이다. 쉘이 대화형으로 실행할 때 말단으로부터 지령을 읽어 들이고 그 지령행을 단어들로 나눈다. 지령행은 공백(빈칸이나 타브)으로 분리되는 한개이상의 단어들(또는 통표들)로 이루어 지고 Enter건을 누를 때 발생하는 행바꾸기문자로 끝난다. 첫번째 단어는 지령이고 그다음 단어는 지령의 추가선택나 인수들이다. 지령은 ls나 pwd와 같은 UNIX실행형 프로그람, 별명, cd나 jobs와 같은 내부지령 혹은 쉘스크립트들이다. 지령은 메타문자라는 특수문자를 포함할수 있는데 쉘은 그것을 지령행을 구문해석할 때 해석해야 한다. 지령행에 있는 마지막문자가 행바꾸기문자가 뒤따르는 거꿀빗선이면 행은 다음행으로 계속될수 있다.[6]

탈퇴상태와 printexitvalue변수 지령이나 프로그람이 끝나면 그것은 어미프로쎄스에 탈퇴상태를 되돌린다. 탈퇴상태는 0~255사이의 수자이다. 습관상 프로그람이 탈퇴할 때 귀환된 상태가 0이면 프로그람이 실행에서 성공한것이다. 탈퇴상태가 0이 아니면 그것은 실패한것이다. 만일 프로그람이 비정상적으로 끝났으면 0200이 상태에 추가된다. 실패한 내부지령들은 탈퇴상태로 1을 되돌린다. 성공하면 0탈퇴상태를 되돌린다.

tcsh상태변수 또는 ?변수는 실행된 마지막지령의 탈퇴상태의 값으로 설정된다. 프로그람의 성공이나 실패는 프로그람을 작성한 작성자에게 달려 있다. tcsh변수 printexitvalue를 설정함으로써 프로그람이 령이 아닌 값으로 탈퇴하였을 때 그의 상태가 자동적으로 인쇄된다.

실례 13-9

1. > **grep "ellie" /etc/passwd**
   ellie:GgMyBsSJavd16s:501:40:E Quigley:/home/jody/ellie: /bin/tcsh
2. > **echo $status** or **echo $?**
   0
3. > **grep "nicky" /etc/passwd**
4. > **echo $status**
   1
5. > **grep "scott" /etc/passsswd**
   grep: /etc/passsswd: No such file or directory
6. > **echo $status**
   2
7. > **set printexitvalue**
   > **grep "xxx" /etc/passwd**
   Exit 1

---

6. 지령행의 길이는 256 문자이상이다.

설명

1. grep프로그람은 /etc/passwd파일에 있는 패턴 ellie를 찾고 성공한다. /etc/passwd로부터 행이 현시된다.
2. status변수는 grep지령의 탈퇴값으로 설정된다. 0은 성공을 의미한다. 변수는 또한 탈퇴상태를 가진다. 이것은 탈퇴상태를 검사하기 위하여 bash와 ksh쉘들에 의해 리용되는 변수이다(csh는 리용하지 않는다.).
3. grep프로그람은 /etc/passwd파일에서 사용자 mickey를 찾을수 없다.
4. grep프로그람은 패턴을 찾을수 없으므로 1탈퇴상태를 복귀한다.
5. grep는 파일 /etc/passwd를 열수 없으므로 실패한다.
6. grep는 파일을 찾을수 없으므로 탈퇴상태 2를 되돌린다.
7. 특수한 tcsh변수 7.printexitvalue가 설정된다. 그것은 령이 아닌 값으로 탈퇴하는 임의의 지령의 탈퇴값을 자동적으로 인쇄한다.

지령그룹화 지령행은 다중지령으로 이루어 질수 있다. 매 지령은 반두점에 의해 구분되며 지령행은 행바꾸기문자로 끝난다.

실례 13-10

```
> ls ; pwd;   cal 2001
```

설명

지령들은 행바꾸기문자가 나타날 때까지 왼쪽에서 오른쪽으로 실행된다.

지령들은 모든 출력이 또 다른 지령에 파이프되거나 파일에로 방향바꾸기되도록 그룹화할수 있다. 쉘은 자식쉘안에서 지령들을 실행한다.

실례 13-11

```
1. >  (ls ; pwd; cal 2001 ) > outputfile
2. > pwd; ( cd / ; pwd ) ; pwd
   /home/jody/ellie
   /
   /home/jody/ellie
```

설명

1. 이 지령들의 출력은 outputfile이라고 불리우는 파일에 보내진다. 괄호가 없이 첫 2개 지령들의 출력을 현시장치에 보내며 cal지령의 출력이 output파일에 방향바꾸기한다.
2. pwd지령은 현재작업등록부를 표시한다. 괄호는 그안에 있는 지령들이 자식쉘에 의해 처리되도록 한다. cd지령은 쉘에 내장되여 있다. 자식쉘안에 있는 동안 등록부는 root로 변경되고 현재작업등록부가 현시된다. 자식쉘밖에 있을 때 초기쉘의 현재작업등록부가 현시된다.

지령들의 조건실행 2개의 지령문자렬들이 조건실행할 때 2개의 특수메타문자들인 2

개의 앰퍼샌드(&&) 또는 2중수직선(||)에 의해 분리된다. 이 메타문자들중의 어느 하나의 오른쪽에 있는 지령은 왼쪽에 있는 지령의 탈퇴조건에 기초하여 실행되거나 되지 않게 된다.

실례 13-12

> **grep ' ^tom : ' \ etc\ passwd && mail tom <letter**

설명

첫 지령이 성공하면 &&다음의 두번째 지령이 실행된다. grep지령이 passwd파일에서 tom을 성과적으로 찾으면 오른쪽 지령이 실행된다. mail프로그람은 tom에 letter파일의 내용을 보낸다.

실례 13-13

> **grep ' ^tom: ' \ etc\ passwd \ echo "tom is not a user here."**

설명

첫 지령이 실패하면 ||뒤에 있는 두번째 지령이 실행된다. grep지령을 passwd파일에서 tom을 찾지 못하면 오른쪽 지령이 실행된다. echo프로그람은 현시장치에 tom이 사용자가 아니라고 인쇄한다.

배경에 있는 지령 일반적으로 지령을 실행할 때에 그것은 전경에서 실행되며 재촉문은 지령이 실행을 완성할 때까지 다시 나타나지 않는다. 지령이 완성되기를 기다리는 것이 언제나 편리한것은 아니다. 쉘재촉문은 지령행의 끝에 &를 써넣으면 즉시에 복귀되여 마지막지령을 완성함으로써 다른 지령을 시작할수 있게 한다. 배경에서 실행되는 지령을 배경일감이라고 하는데 그 출력은 그 일감을 처리할 때 현시장치로 간다. 2개 지령을 동시에 현시장치에 출력을 보내면 혼란이 일어 날수 있다. 혼란을 피하기 위하여 배경에서 실행하는 작업의 출력을 파일로 보내거나 그것을 인쇄기와 같은 다른 장치에 파이프할수 있다. 배경안에서 새로운 쉘창문을 시작하는것은 편리하다. 이때 시작한 창문과 새로운 쉘창문에 다 접근한다.

실례 13-14

1. > **man tcsh \ lpr &**
2. *[1] 4664*
3. >

설명

1. tcsh프로그람을 위한 사용지도서페지들로부터 출력을 인쇄기에 파이프한다. 지령행의 끝에 있는 &는 배경에 일감을 놓는다.
2. 현시장치에 표시되는 두 수자가 있다. 중괄호안에 있는 수자는 배경에 배치되는 첫 일감임을 가리킨다. 두번째 수자는 이 일감의 PID이다.
3. 쉘재촉문이 즉시에 나타난다. 프로그람이 배경에서 실행되고 있을 때 쉘은 전경에서 다른 지령을 입력할것을 요구한다.

제 3 절. 지령행지름길

1. 리력

history기구는 TC쉘에 내장되여 있다. 그것은 기억기에 지령행에서 입력한 events라고 하는 지령들을 련속적으로 번호가 붙은 목록으로 기억한다. 리력사건의 말단에서 입력된 사건이 시간이 지나간 다음에 기억된다. 리력사건의 번호는 추가된다. 쉘이 말단으로부터 지령을 읽을 때 그것은 지령행을 단어들로 가르며(공백을 사용하여 단어를 분리한다.) 리력목록에 행을 기억하고 해석한 다음 실행한다. 이전에 입력된 지령은 항상 기억된다. 리력목록에서 지령은 임의의 시각에 다시 호출할 때 다시 입력하지 않고도 재실행할수 있다. 등록가입대화조종기간에 입력한 지령들은 탈퇴할 때까지 리력목록에 추가되는데 이때 그것들은 .history라고 부르는 홈등록부에 있는 파일에 기억할수 있다.[7] 리력목록과 리력파일이라는 말을 혼돈할수 있다. 리력목록은 쉘의 기억기에 현재 있는 지령행들로 이루어 진다. .history라고 불리우는 리력파일은 지령들이 앞으로 리용하기 위하여 기억하는 본문파일이다. 내부변수 savehist는 등록탈퇴할 때 .history파일에 리력목록을 기억하고 시동할 때 기억기에 그 내용을 적재한다(표 13-2에서 history지령에 대한 –S와 –L추가선택들을 보여 준다.). history내부지령은 리력목록을 현시한다. 그것은 일련의 인수들을 지원하여 리력이 표시되는 방식을 조종한다.

표 13-2. history지령과 추가선택

추가선택	의 미
-h	수자가 없는 리력목록을 인쇄한다.
-T	설명문형식으로 timestamps를 인쇄한다.
-r	리력목록을 거꾸로 인쇄한다.
-S[filename]	.history 또는 filename이 주어 지면 리력목록에 기억된다.
-L[filename]	리력파일(.history 또는 filename)을 리력목록에 추가한다.
-M[filename]	현재리력목록을 리력파일의 내용과 합치는것을 제외 하고는 –L과 같다.
-c	기억기에서 리력목록을 지우고 리력파일은 지우지 않는다.
N	n은 번호이다. 실례로 history5. 이것은 현시되는 행들의 개수를 조종한다.

비록 리력파일의 기정이름이 .history일지라도 그 이름은 내부변수 histfile에 다른 이름을 대입하여 변경할수 있다.

history쉘변수는 몇개의 지령이 표시되는가를 지정하는 수자로 설정되며 histdup변수가 설정되여 중복되는 입력이 리력파일에 추가되지 않도록 할수 있다.

실례 13-15

```
(The Command Line)
 > history
1.   17:12    cd
```

[7]. history 파일의 이름은 histfile 쉘변수에 새로운 이름을 대입하여 변경시킬수 있다.

2. 17:13 ls
3. 17:13 more /etc/fstab
4. 17:24 /etc/mount
5. 17:54 sort index
6. 17:56 vi index

설명

리력목록은 지령행에 입력된 마지막지령들을 표시한다. 목록에 있는 매 사건앞에 한개 수자(사건번호라고 부른다.)와 그것이 지령행에 입력된 시간이 붙는다.

history변수 TC쉘 history변수를 말단에 표시될 리력목록의 사건들의 개수로 설정할수 있다. 일반적으로 이것은 사용자의 초기화파일인 /etc/.cshrc 또는 ~/.tcshrc파일에서 설정한다. 암시적으로 100으로 설정한다. 리력을 형식화하는 방법을 조종하기 위해 리력변수에 다른 값을 선택할수도 있다. 이 값은 재촉문변수와 같은 형식화문자렬을 리용한다. history를 위한 기정형식문자렬은 %h\ t%T\ t%R\ n이다.

실례 13-16

1. **set history=1000**
2. **set history= (1000 '%B%h %Rn')**
3. **history**
   136 history
   137 set history = (1000 '%B%h %R\ n')
   138 history
   139 ls
   140 pwd
   141 cal
   141 pwd
   142 cd

설명

1. 말단에서 입력된 마지막 1,000개의 지령들은 history지령을 입력함으로써 현시장치에 현시할수 있다.
2. 말단에서 입력된 마지막 1,000개의 지령들을 현시한다. 형식문자렬은 먼저 사건번호(%h)가 오고 그다음에 공백, 마지막에 행바꾸기문자(\ n)가 뒤따르는 지령행에 입력된 지령(%R)이 리력목록에 놓이는 굵은 본문(%B)으로 표시되도록 한다.
3. history를 입력할 때 새로운 형식이 보여 진다. 이것은 다만 실지 리력목록의 선택된 부분이다.

리력보관과 savehist변수 등록가입할 때 리력사건들을 보존하기 위하여 savehist변수를 설정한다. 이 변수는 보통 사용자의 초기화파일인 .tcshrc파일에서 설정된다. savehist에 대입된 첫번째 값이 수값이라면 history변수가 설정된 경우 그 값은 history

변수에 설정된 수값을 초과할수 없다. 두번째 값이 merge로 설정되면 리력목록은 존재하는 리력파일을 교체하지 않고 그것과 합쳐 진다. 그것은 timestamp에 의해 정돈되며 가장 최근의 사건들을 기억한다.

실례 13-17

1. **set savehist**
2. **set savehist =1000**
3. **set savehist =1000 merge**

설명

1. 리력목록으로부터 지령들은 리력파일안에 기억되며 등록가입한 다음에 리력목록의 꼭대기에 놓인다.
2. 리력파일을 리력목록에 있는 마지막 1,000개의 지령으로 교체하고 기억한다. 그것을 다음에 등록가입할 때 현시한다.
3. 현존 리력파일을 교체하지 않고 현재리력목록을 등록탈퇴할 때 존재하는 리력파일과 합쳐 져 등록가입후에 기억기에 적재한다.

리력현시 history지령은 리력목록에 있는 사건들을 표시한다. history지령은 또한 사건들의 번호와 표시될 사건들의 형식을 조종하는 추가선택들을 가지고 있다. 사건들의 번호붙이기는 반드시 1에서 시작하는것은 아니다. 만일 리력목록에 100개 지령들을 가지고 있고 리력변수를 25로 설정하였다면 마지막에 기억된 25개 지령만을 보게 된다.

실례 13-18

1. **> set history = 10**
2. **> history**

   1 ls
   2 vi file1
   3 df
   4 ps -eaf
   5 history
   6 more /etc/passwd
   7 cd
   8 echo $USER
   9 set
   10 ls

설명

1. 리력변수를 10으로 설정한다. 10개이상의 행들이 있다고 해도 리력목록의 마지막 10개 행만을 표시한다.
2. 리력으로부터 마지막의 10개 사건들을 표시한다. 매 지령에 번호를 붙인다.

실례 13-19

1. **> history –h** *# Print without line numbers*
   ls
   vi filel
   df
   ps -eaf
   history
   more /etc/passwd
   cd
   echo $USER
   set
   history -n

2 **> history –c**

설명

1. h추가선택을 리용하면 리력목록을 행번호없이 표시한다.
2. C추가선택을 리용하면 리력목록을 지운다.

실례 13-20

> history –r *# Print the history list in rverse*
11 history -r
10 history -h
9 set
8 echo $USER
7 cd
6 more /etc/passwd
5 history
4 ps –eaf
3 df
2 vi filel
1 ls

설명

리력목록은 반대순서로 현시된다.

실례 13-21

> history 5 *# Print the last 5 events on the history list*
7 echo $USER
8 cd
9 set
10 history -n

11 history 5
설명
마지막 5개 지령이 리력목록에서 실행된다.

리력파일로부터 지령들의 호출 리력목록으로부터 지령들을 호출하고 반복시키는 몇가지 방법이 있다. 방향건을 리용하여 리력목록을 우아래로, 좌우로 이동시켜 편집한다. 리력바꾸어넣기라고 하는 도구를 리용하여 맞춤법오유들을 다시 실행하여 정정한다. 또는 내부 emacs나 vi편집기를 리용하여 이전의 지령들을 회복하고 편집실행한다. 이러한 방법들가운데서 제일 좋은 방법을 선택한다.

1. 방향건

   리력목록에서 지령들을 호출하기 위해 건반우에 있는 방향건을 리용하여 리력목록을 좌우, 상하로 이동한다. 또한 리력목록의 임의의 행을 지우기나 뒤걸음 등에 대한 표준건을 리용하여 편집한다. 행편집을 끝내자마자 행복귀(Enter)건을 누르면 지령행이 재실행된다. 또한 표준 emacs나 vi지령을 리용하여 리력목록을 편집한다(표 13-5와 13-6 참고). 방향건은 vi와 emacs건맺기에 대하여 같은 방법으로 동작한다.

표 13-3. 방향건

↑	웃방향건은 리력목록을 우로 이동시킨다.
↓	아래방향건은 리력목록을 아래로 이동시킨다.
→	오른쪽방향건은 유표를 history지령의 오른쪽으로 이동시킨다.
←	왼쪽방향건은 유표를 history지령의 왼쪽으로 이동시킨다.

2. 재실행과 감탄부호

   리력목록에서 지령을 재실행하기 위해 감탄표(bang)를 리용하여 리력바꿔넣기를 시작한다. 감탄부호는 행의 어디서나 시작할수 있으며 거꿀빗선을 리용하여 특수문자로 리용되지 않게 한다. !다음에 공백, 타브, 또는 행바꾸기문자가 놓이면 그것은 해석되지 않는다. 리력바꿔넣기의 여러가지 방법들을 사용하여 리력목록의 어느 부분을 다시 실행하려는가를 지정할수 있다(표 13-4에서 보여 준다.). 2개의 감탄부호를 입력하면 마지막지령이 재실행된다. 감탄부호다음에 한개의 수를 입력하면 그 수는 리력목록의 지령과 련관되여 그 지령을 실행한다. 감탄부호와 한개 문자를 입력하면 그 문자로 시작되는 마지막지령을 실행한다. 탈자기호(^)도 이전의 지령을 편집하기 위한 지름방식으로 리용한다.

   리력바꿔넣기가 진행된후 리력목록을 지령에서 보여 준 바꿔넣기결과로 갱신한다. 실례로 !!를 입력하면 마지막지령을 재실행하여 확장된 형식으로 리력목록에 기억한다. 마지막지령을 문자형식으로 리력목록에 추가하려면 histlit 쉘변수를 설정한다.

실례 13-22

```
1.  > date
     Mon Feb   8 12:27:35 PST 2001
2.  >   ! !
     date
     Mon Aug   10 12:28:25 PST 2001

3.  > !3
     date
     Mon Aug   10 12:29:26 PST 2001
4.  > id
    date
    Mon Aug   10 12:30:09 PST 2001
5.  > dare
     dare: Command not found.
6.        >   ^r^t
    date
    Mon Apr   10 16:15:25 PDT 2001
7.  > history
     1   16:16   ls
     2   16:16   date
     3   16:17   date
     4   16:18   date
     5   16:18   dare
     6   16:18   date
8.  > set histlit
9.  > history
     1   16:18   ls
     2   16:19   date
     3   16:19   !!
     4   16:20   !3
     5   16:21   dare
     6   16:21   ^r^t
```

설명

1. UNIX data지령을 지령행에서 실행한다. 리력목록을 갱신한다. 이것은 목록우의 마지막지령이다.
2. !!는 리력목록으로부터 마지막지령을 얻는다. 그 지령을 재실행한다.
3. 리력목록에 있는 세번째 지령을 재실행한다.
4. 문자 d로 시작되는 리력목록의 마지막지령을 재실행한다.
5. 지령이 잘못 입력되였다.
6. ^기호는 리력목록에서 마지막지령으로부터 문자렬을 바꾸어 넣는데 리용한다. 첫번째 나타나는 r가 t로 교체된다.
7. 리력바꿔넣기가 진행된후에 history지령은 리력목록을 표시한다.
8. histlit를 설정하여 쉘은 리력바꿔넣기를 수행하지만 리력목록에 그것이 입

력된 그대로 놓는다.

9. histlit를 설정할 때 history지령의 출력은 리력바꿔넣기가 진행되기전에 문자로 입력된 지령들을 보여 준다(이것이 바로 demo이다. 리력번호는 정확치 않다.).

실례 13-23

1. > **cat filel file2 file3**
   <Contents of files displayed here>
   > **vi ! :1**
   vi fi 1 el
2. > **cat filel file2 file3**
   <Contents of file, file2, *and* file3 are displayed here
   > **ls ! :2**
   ls file2
   file2
3. > **cat filel file2 file3**
   > **ls !:3**
   ls file3
   file3
4. > **echo a b c**
   a b c
   > echo !$
   echo c
   c
5 > **echo a b c**
   *a b c > echo !^*
   echo a
   a
6. > **echo a b c**
   a b c
   *> echo !**
   echo a b c
   a b c
7. > **!! :p**
   echo a b c

설명

1. cat지령은 현시장치에 file1, file2, file3의 내용을 표시한다. 리력목록을 갱신한다. 지령행을 단어번호가 령으로 시작되는 단어들로 가른다. 단어번호앞에 옹근두점이 있으면 그 단어를 리력목록에서 추출할수 있다. !:1표기의 의미는 다음과 같다. 리력목록에 있는 마지막지령으로부터 첫번째 인수를 얻고 그것을 지령문자렬에서 교체한다. 마지막지령의 첫 인수는 file1이다(단어 0은 지령 그자체이다.).
2. !:2는 마지막지령 file2의 두번째 인수로 교체되여 인쇄되는 ls.file2에 대한

인수로서 주어 진다(file2는 세번째 단어이다.).

3. ls !:3은 다음과 같다. 리력목록에 있는 마지막지령에 가서 네번째 단어를 얻고(단어는 0에서 시작한다.) 그것을 인수로서 ls지령에 보낸다(file3은 네번째 단어이다.).
4. $기호를 가진 감탄부호(!)는 리력목록의 마지막지령의 마지막인수를 표시한다. 마지막인수는 c이다.
5. 탈자기호(^)는 지령다음에 첫번째 인수를 표시한다. ^를 가진 감탄부호(!)는 리력목록의 마지막지령의 첫 인수를 표시한다. 마지막지령의 첫 인수는 a이다.
6. 별표식(*)은 지령뒤에 모든 인수를 나타낸다. *를 가진 감탄부호(!)는 리력목록의 마지막지령의 모든 인수를 표시한다.
7. 리력목록으로부터 마지막지령은 인쇄되지만 실행되지 않는다. 리력목록이 갱신된다. 지금 그 행에서 탈자기호바꿔넣기를 진행한다.

표 13-4. 바꿔넣기와 리력

사건지적자	의 미
!	리력바꿔넣기의 시작을 나타낸다.
!!	이전 지령을 재실행한다.
!N	리력목록에서 N번째 지령을 재실행한다.
!-N	현재지령에서 N번째 뒤에 놓인 지령을 재실행한다.
!string	문자렬로 시작되는 마지막지령을 재실행한다.
!? string?	문자렬을 포함하는 마지막지령을 재실행한다.
!? string?%	문자렬을 포함하는 리력목록으로부터 가장 최근의 지령행인수를 재실행한다.
!^	현재지령행에서 마지막 history지령의 첫번째 인수를 리용한다.
!*	현재 지령행에서 마지막 history지령의 모든 인수들을 리용한다.
!$	현재지령행에서 마지막 history지령의 마지막인수를 리용한다.
!!string	이전의 지령물음표를 추가하고 실행시킨다.
!Nstring	리력목록에 있는 N번째 지령에 문자렬을 추가하고 실행시킨다.
!N:s/old/new/	이전의 N번째 지령에서 이전의 물음표들을 새로운 문자렬로 전체대역에서 바꾸어 넣는다.
^old^new^	마지막 history지령에서 낡은 문자렬을 행바꾸기문자렬로 교체한다.
Command!N:wn	이전의 N번째 지령으로부터 인수(wn)를 추가하는 현재지령을 실행시킨다. wn은 0, 1, 2…에서 시작하는 수인데 이전의 지령으로부터 단어들의 번호를 표시한다. 단어 0은 지령 그자체이고 1은 첫번째 인수이다.
!N:p	리력목록의 밑에 지령을 넣고 그것을 인쇄하지만 실행시키지는 않는다.

2. 내부지령행편집기

지령행을 emacs나 vi편집기에서 리용하는것과 같은 형태의 건문자렬을 리용하여 편집할수 있다. 편집기지령을 리용하여 리력목록을 우아래로 이동시킬수 있다. 일단 지령을 찾으면 그것을 편집할수 있고 Enter건을 눌러서 재실행시킬수 있다. 쉘을 번역할 때 그것은 emacs편집기를 위한 건맺기기정모임을 준다.

bindkey내부지령 내부 bindkey지령은 지령행편집을 위한 vi나 emacs를 선택하는데 리용하며 매개 편집기를 위한 건맺기를 표시하고 설정하는데 리용한다. 지령편집기로서 vi를 리용하기 위하여 –v추가선택을 가진 bindkey를 리용한다.

bindkey –v

그리고 emacs로 되돌아 간다.

bindkey –e

편집기지령들의 목록과 개개가 무엇을 하는가를 간단히 설명하기 위하여 다음과 같이 입력한다.

bindkey –l

실제의 건과 지령들이 어떻게 맺어지는가를 보기 위하여

bindkey

라고 입력한다.

실지로 건들을 지령에 맺어 주기 위하여 773페지의 《맺기건》을 참고한다.

vi내부편집기 리력목록을 편집하기 위하여 지령행에 가서 Esc건을 누른다. 다음에 표준 vi이동건과 같이 리력목록에서 웃쪽으로 이동하려면 K건을 누르고 아래 방향으로 움직이려면 J건을 누른다. 지령을 편집하려고 할 때 본문의 좌우이동, 지우기, 삽입,변경을 위해 vi에서 리용하는 표준건을 리용한다. vi지령에 대하여 표 13-5에 주었다. 편집한 후에 Enter건을 누른다. 지령이 실행되고 리력목록의 마지막에 추가된다. 만일 본문을 추가 혹은 삽입하려면 삽입지령들가운데서 임의의것(i, e, o, O 등)을 리용한다. Vi는 2개 방식 즉 지령방식과 삽입방식을 가지고 있다는것을 알고 있어야 한다. 실지로 본문을 입력하고 있을 때에는 언제나 삽입방식에 있다. 지령방식으로 돌아 가기 위해서는 탈퇴건(Esc)을 누른다.

표 13-5. vi지령

지 령	기 능
리력파일을 이동시키기	
Esc k 또는 +	리력목록을 우로 이동시킨다.
Esc j또는 -	리력목록을 아래로 이동시킨다.
G	리력파일에서 첫번째 행으로 이동시킨다.
5G	리력파일에서 다섯번째 지령으로 이동시킨다.
/string	리력파일을 우로 탐색한다.

(표계속)

지 령	기 능0
?	문자렬을 리력파일아래로 탐색한다.
한 행에서 이동	
H	한 행에서 왼쪽으로 이동한다.
L	한 행에서 오른쪽으로 이동한다.
B	한 단어를 뒤로 이동한다.
e or w	한 단어를 앞으로 이동한다.
ˆ or 0	행의 첫번째 문자의 시작으로 이동한다.
$	행의 끝으로 이동한다.
vi로 편집	
a A	본문을 추가한다.
i I	본문을 삽입한다.
dd dw x	본문을 지워 완충기에로 넣는다(행, 단어 또는 문자).
cc C	본문을 변경시킨다.
u U	본래대로 돌아가기
yy Y	완충기에 행을 복사한다.
PP	완충기에 복사되거나 지워진 행의 아래 또는 우에 넣는다.
RR	행우에서 한개 문자나 임의의 본문을 교체한다.

emacs내부편집기 만일 vi와 같은 emacs내부편집기를 리용하고 있다면 지령행에서 시작한다. 리력파일의 우로 이동하려면 ˆP를 누른다. 아래로 이동하기 위하여서는 ˆN을 누른다. 본문을 변경하고 수정하기 위하여 emacs편집지령을 리용하고 Enter건을 누르면 지령을 재실행한다. 표 13-6에 emacs지령들을 주었다.

표 13-6. emacs지령

지 령	기 능
Ctrl-P	리력파일을 우로 이동시킨다.
Ctrl-N	리력파일을 아래로 이동시킨다.
Ctrl-B	한 문자를 뒤로 이동시킨다.
Ctrl-R	문자렬을 뒤방향으로 탐색한다.
Ctrl- F	한 문자를 앞으로 이동시킨다.
Esc B	한 단어를 뒤로 이동시킨다.
Esc F	한 단어를 앞으로 이동시킨다.
Ctrl-A	행의 시작으로 이동시킨다.

(표계속)

지 령	기 능
Ctrl-E	행의 끝으로 이동시킨다.
Esc<	리력파일의 첫번째 행으로 이동시킨다.
Esc>	리력파일의 마지막행으로 이동시킨다.
emacs로 편집하기	
Ctrl-U	행을 지운다.
Ctrl-Y	행을 다시 복귀한다.
Ctrl-K	유표로부터 끝행까지 지운다.
Ctrl-D	한 문자를 지운다.
Esc D	한 단어를 앞으로 지운다.
Esc H	한 단어를 뒤로 지운다.
Esc space	유표위치에서 표식을 설정한다.
Ctrl-X Ctrl-X	유표와 표식을 변화시킨다.
Ctrl-P Ctrl –Y	유표로부터 표식까지의 구역을 완충기에 보내여(Ctrl-P) 그것을 복사한다(Ctrl-Y).

맺기건 bindkey내부지령은 emacs와 vi를 위한 건맺기를 포함하여 모든 표준건맺기를 표시한다. 건맺기들은 4개 그룹으로 나누어 진다. 즉 표준건맺기, 대리건맺기, 다중문자건맺기, 방향건맺기이다. bindkey지령은 또한 현재건맺기들을 변경할수 있게 한다.

실례 13-24

```
1.  > bindkey
Standard   key   bindings
"^@"              ->   is undefined
"^A"              ->   beginning-of line
"^B"              ->   backward-char
"^C"              ->   tty-sigintr
"^D"              ->   list-or-eof
"^E"              ->   end-of-line
"^F"              ->   forward-char
"^L"              ->   clear-screen
"^M"              ->   newline
        ...        ...
Alternative   key   bindings
"^@"              ->   is undefined
"^A"              ->   beginning-of-line
"^B"              ->   is undefined
```

```
"^C"        ->  tty-sigintr
"^D"        ->  list-choices
"^E"        ->  end-of-line
"^F"        ->  is undefined
      ......    .....
Multi-character  bindings
"^[[A"      -> up-history
"^[[B"      -> down-history
"^[[C"      -> forward-char
"^[[D"      -> backward-char
"^[[OA"     -> up-history
"^[[OB"     -> down-history
      ...  ......
Arrow key  bindings
Down        -> down-history
Up          -> up-history
Left        ->backward-char
Right       -> forward-char
```

bindkey에 대한 –1추가선택은 편집기지령들과 그것들이 무엇을 하는가를 표시한다. 실례 13-25에 보여 주었다.

실례 13-25

```
> bindkey   -1
backward-char
    Move back a character
backward-delete-char
    Delete the character  behind cursor
backward-delete-word
    Cut from beginning of current word to cursor – saved in cut buffer
backward-kill-line
    Cut from beginning of line to cursor – saved in cut buffer
backward-word
    Move to beginning of current word
beginning-of-line
    Move to beginning of line
capitalize-word
    Capitalize the characters from cursor to end of current word
change-case
    Vi change case of character under cursor and advance one character
change-till-end-of-line
    Vi change to end of line
clear-screen
  Standard key bindings
```

bindkey지령은 또한 실례 13-26에서 보여 준것처럼 개별적인 건맺기들에 대한 값을 현시한다. 기정으로 emacs넘기기를 보여 주지만 bindkey에 대한 –a추가선택을 리용하면 vi건들에 대한 대리넘기기를 현시한다. bindkey에 대한 인수는 특수문자렬들로 지정되여 건이 맺어 지는 편집지령건다음에 놓이는 건문자렬들을 표시한다. 건은 emacs나 vi 편집기지령뿐아니라 UNIX지령과 문자렬들도 맺어 줄수 있다.

표 13-7. 건맺기문자

메타문자	의 미
ˆC	Control-C
ˆ[	Esc
ˆ?	Del
\ a	Control-G(bell)
\ b	Control-H
\ c	Esc(escape)
\ f	Formfeed
\ n	Newline
\ r	Return
\ t	Tab
\ v	Control-K
\ nnn	ASCII octal number

실례 13-26

1. > **bindkey *L**
   "^L" -> clear-screen
2. > **bindkey "C**
   "^C" -> tty-sigintr
3. > **bindkey "j"**
   "j" -> self-insert-command
4. > **bindkey -v**
5. > **bindkey -a "j"**
   "j" -> down-history

설명

1. control건(ˆL)을 가진 bindkey지령은 control건이 어떤 지령과 맺어 지는가를 표시한다. ˆL은 화면이 지워 지게 한다.
2. Ctrl-C(ˆC)는 일반적으로 프로쎄스를 끝내는 새치기신호와 맺어 진다.
3. 소문자 j는 아무것도 하지는 않지만 그 문자를 완충기에 삽입하는 emacs의

자체삽입지령이다.
4. 대리 vi건맺기를 보기 위하여 여기에서 보여 준것처럼 bindkey-v를 가진 vi지령행편집기를 설정하였다는것을 확인한다.
5. –a추가선택으로 bindkey는 j와 대응하는 대리건을 현시한다. 즉 리력목록을 아래로 움직이기 위한 vi건을 현시한다.

실례 13-27

1. > **bindkey "^T" clear-screen**
2. > **bindkey "^T"**
   "^T" -> clear-screen
3. > **bindkey -a -"T"**
   "^T" -> undefined-key
4. > **bindkey -a** ***[Coutrol-v Control t] clear-screen***
   Press keys one after the other
5. > **bindkey -a [Control-v Control t]**
   "^T" -> clear-screen
6. > **bindkey -s '\ ehi' 'Hello to you! \ n'**
   > echo [Esc]hi Press escape followed by 'h' and 'i'
   Hello to you!
   >
7. > **bindkey '^[hi'**
   "^[hi" -> "Hello to you!"
8. > **bindkey -r '\ [hi'**
9. > **bindkey '\ ehi**
   Unbound extended key "^hi"
10. > **bindkey -c '\ ex' 'ls | more'**

설명

1. Ctrl-T는 화면을 지우기 위한 지령인 기정적인 emacs건넘기기와 맺어 진다. 이 건문자렬은 초기에 아무것과도 맺어 지지 않는다. control건과 T건을 같이 누를 때 화면은 지워 진다.
2. bindkey지령은 인수로서 건문자렬을 가질 때 만약 하나라면 그 문자렬을 위한 넘기기를 현시한다.
3. –a추가선택과 건문자렬을 가지고 있는 bindkey는 대리건넘기기인 vi의 값을 현시한다. 이 실례에서 –a추가선택과 건문자렬을 가진 bindkey는 대리넘기기(vi)가 이 문자렬의 어떤것과도 맺어 지지 않도록 한다는것을 보여 준다.
4. –a추가선택이 있는 bindkey는 건을 대리건과 대응시켜 vi와 맺어 줄수 있다. Control-V를 누르고 그다음에 Control-T를 누르면 건문자렬이 만들어지고 값 clear-screen이 대입된다. Ctrl-V/ctrl-T는 이전 실례에서 보여 준

것처럼 “^T”라고 표시할수 있다.

5. bindkey함수는 ^T와 그의 지령을 위한 대리넘기기를 현시한다.
6. –s지령을 가진 bindkey는 문자렬을 건문자렬에 맺어 준다. 여기서 문자렬 Hello to you!\ n은 탈출문자 hi와 맺어 진다다. Esc건을 누르고 다음에 h와 i건을 누르면 문자렬을 표준출력에로 보낸다.
7. bindkey지령은 탈출문자렬 hi를 위한 맺기를 현시한다. ^[는 Esc를 나타내는 또 다른 방법이다.
8. –r추가선택과 함께 bindkey는 건맺기를 지우기한다.
9. 건맺기가 제거되였기때문에 출력은 이 확장된 건문자렬이 맺어 지지 않는다는것을 알려 준다.
10. –c추가선택과 함께 bindkey는 건문자렬을 UNIX지령에 맺어 줄수 있다. 이 실례에서 Escape건을 누르고 X건을 누르면 지령 ls가 more에 파이프된다.

표 13-8. bindkey추가선택

bindkey추가선택	기 능
bindkey –a	대리건넘기기를 허용한다.
bindkey –d	기정맺기들을 회복한다.
bindkey –e	emacs맺기들을 리용한다.
bindkey –l	모든 편집지령들과 그것들의 의미를 현시한다.
bindkey –u	사용법통보문을 현시한다.
bindkey –v	vi건맺기들을 리용한다.
bindkey key	건에 대한 맺기를 현시한다.
bindkey key command	건을 emacs나 vi지령과 맺어 준다.
bindkey -c key command	건을 UNIX지령과 맺어 준다.
bindkey -s key string	건을 문자렬과 맺어 준다.
bindkey -r key	c 건맺기를 지운다.

3. 지령, 파일이름, 변수의 완성

입력을 기억하기 위하여 tcsh는 지령, 파일이름, 변수의 일부분을 입력하고 타브건을 눌러서 단어의 나머지부분을 완성할수 있게 하는 완성기구를 가지고 있다. 만일 어떤 지령의 첫 몇개 문자를 입력하고 타브건을 누르면 tcsh는 지령이름을 완성하기위해 시도한다. 만일 tcsh가 지령이 존재하지 않아서 그것을 완성할수 없다면 말단은 경고를 보내고 유표는 지령끝에 남아 있는다. 어떤 문자들로 시작되는 한개이상의 지령들이 있다면 Ctrl-D를 눌러서 그 문자렬들로 시작되는 모든 지령들을 표시한다. 파일이름과 변수완성은 지령완성과 같다. 파일이름완성을 리용하면 같은 문자를 가지고 시작하는 여러개의 파일들이 있을 때 tcsh는 대조하는 가장 짧은 이름을 완성하고 문자가 같지 않을 때까지 파일이름을 확장한 다음 그 나머지를 완성하도록 유표를 깜빡이게 한다(실례 13-28).

atolist변수 autolist변수가 설정되여 가능한 완성들의 수가 있다면 타브건을 눌렀을 때 수행하고 있는 완성의 형태에 따르는 가능한 모든 지령, 변수들, 파일이름들이 표시된다.

실례 13-28

```
1.  > ls
    filel file2 foo foobarckle fumble
2.  > ls fu[tab]          # eExpands to filename to fumble
3.  > ls fx[tab]          # Terminal beeps, nothing happens
4.  > ls fi[tab]          # Expands to file_ (_ is a cursor)
5.  > set autolist
6.  > ls f[tab]           # Lists all possibilities
    filel file2 foo foobarckle fumble
7.  > ls foob[tab]        # Expands to foobarckle
8.  > da[tab]             # Completes the date command
    date
    Fri Aug 9 21:15:38 PDT 2001
9.  > ca[tab]             # Lists all commands starting with ca
    cal     captoinfo case     cat
10.  > echo $ho[tab]me    # Expands shell variables
     /home/ellie/
11.  > echo $h[tab]
    history home
```

설명

1. 현재작업등록부에 대한 모든 파일들을 현시한다.
2. fu가 입력된 다음 타브건을 누르면 파일이름을 fumble로 완성하여 표시한다.
3. fx로 시작하는 파일은 없으므로 말단은 경고음을 내고 유표는 그자리에 그대로 있지만 아무것도 하지 않는다.
4. fi로 시작되는 파일들이 여러개 있다. 파일이름을 문자들이 같지 않을 때까지 완성한다. Ctrl-D를 누를 때 그 문자로 된 모든 파일을 표시한다.
5. autolist변수를 설정한다. 타브건을 누를 때 여러개를 선택할수 있다면 autolist는 모든 가능한 선택들을 현시한다.
6. 타브건을 누른다음 f로 시작하는 모든 파일이름들을 인쇄한다.
7. 타브건을 누를 때 파일이름을 foobarckle로 확장한다.
8. 타브건을 da다음에 누를 때 da로 시작하는 지령만이 date지령으로 된다. 지령이름을 확장하고 실행한다.
9. autolist를 설정하였으므로 ca다음에 타브건을 누를 때 ca로 시작하는 모든 지령들을 표시한다. autolist를 설정하지 않으면 Ctrl-D를 입력하여 목록을 본다.
10. 단어에 있는 안내 $기호는 타브건을 누를 때 쉘이 변수확장을 진행하여 단어를 완성하도록 한다. 변수 home을 완성한다.
11. 이 실례에 있는 변수완성은 애매하다. 타브건을 눌러서 완성이 진행될 때 모든 가능한 쉘변수들을 표시한다.

fignore변수 쉘변수 fignore는 파일이름완성을 리용할 때 어떤 파일이름확장자를 무시하도록 설정한다. 실례로 목적파일을 읽을수 없기때문에 .o에서 끝나는 확장된 파일들을 요구하지 않을수 있다. 또는 파일이름들을 확장할 때 사고로 .gif파일들이 지워 지지 않게 할 필요가 있을수 있다. 이를 위해 파일이름확장으로부터 제외되는 파일확장자들의 목록을 fignore변수에 대입한다.

실례 13-29

1. > **ls**
   baby   box.gif   file2   prog.c
   baby.gif   filel   file 3   prog.o

2. > **set fignore = (.o .gif)**

3. > **echo ba***[tab]*   *# Completes baby but ignores baby. gif*
   baby

4. > **echo box***[tab].gif*   *# fignore is ignored if only* ***one***   *box.gif*
   completion   *# -is possible*

5. > **vi** ***prog****[tab]*   *# Expands to prog.c*

   Starts vi with prog.c as its argument

설명

1. 현재작업등록부에 있는 파일들을 표시한다. 일부 파일들은 자기 이름에 확장자를 가지고 있다는데 대해 주의하여야 한다.
2. fignore변수는 파일이름완성을 진행할 때 무시되는 파일이름확장자를 표시하도록 한다. .o나 .gif로 끝나는 모든 파일이 무시된다.
3. 타브건을 누르면 파일 baby만을 표시하고 baby.gif는 표시하지 않는다. .gif파일은 무시된다.
4. 비록 gif를 무시하는 뒤붙이로서 표시하지만 fignore는 3행에서와 같은 .gif확장자를 가지지 않는 파일이름들과 마찬가지로 다른 가능한 완성이 없을 때 영향을 주지 않는다.
5. vi편집기가 호출될 때 prog는 prog.c로 확장된다.

complete쉘변수 이 변수는 많은 기능을 수행하는 변수이다. tcsh 사용지도서페지로부터 할수 있는 모든것에 대하여 아는 것은 좀 어려운 일이지만 여기에 있는 몇가지 실례들은 처음 시작하는데 도움을 줄수 있다. 이것을 리용하여 진행하려는 완성의 종류를 조종할수 있다. 실례로 지령행에서 그 위치에 따라 등록부이름이나 파일이름을 확장하는 완성만을 요구할수도 있고 또는 어떤 지령들을 확장하고 다른것들을 거기에서 제외하도록 하거나 확장할수 있는 가능한 단어들의 목록을 작성할수도 있다. 완성을 리용하여 무엇을 하려고 하든지간에 complete쉘변수는 의심할바없이 쓸모가 있다. 만일 complete쉘변수가 enchance로 설정되면 파일이름완성을 더 세밀하게 진행한다. 이것은 타브완성이 선택을 무시하는 경우 련결부호, 점, 밑선을 단어분리기호 즉 련결부호와 밑선을 같게 취급하게 한다.

실례 13-30

1. **> set complete=enchance**
2. > *ls g..*[tab] **expands to gawk-3.0.3**
   gawk-3.0.3
3. > **ls GAW**[tab] *expands to gawk-3.0.3*
   gawk-3.0.3

설명

1. complete쉘변수를 enchance로 설정하여 타브완성이 선택을 무시하는 경우 련결부호, 점 그리고 밑선을 단어분리기호 즉 련결부호와 밑선을 같은것으로 취급한다.
2. enchance의 설정에서 파일이름완성은 g..을 임의의 부호들앞에 놓이는 한개의 g로 시작되고 2개 부호들(..)과 련결부호, 점 등에 의해서 완성되는 임의의 파일들로 확장된다.
3. enchance를 설정하여 파일이름완성은 GAW를 GAW로 시작하는 임의의 파일들로 확장하는데 여기서 GAW는 소문자와 대문자의 임의의 조합일수 있고 나머지 문자들은 련결부호, 점 그리고 밑선을 비롯해서 임의의 물음표로 될수 있다.

프로그람적완성 완성을 더 많은 특수한 기능들로 전형화하기 위해 완성들을 프로그람화 하여 그것들을 ~/.tcshrc파일에 기억시켜 새로운 TC쉘을 기동할 때마다 그것들이 tcsh환경의 부분으로 되게 한다. 프로그람적완성의 목적은 효률을 높이고 영향을 받는 지령들과 인수들의 형태를 선택하기 위해서이다(단어완성을 위한 타브건과 가능한 모든 완성들을 표시하는 Ctrl-D는 간단한 완성들에서와 같은 방법으로 리용된다.).

완성형태 완성에는 3가지 형태 즉 p, n, c가 있다. p형완성은 위치의존이다. 이것은 지령행에서 단어의 위치에 기초하여 완성을 진행하는 방식을 조종한다. 여기서 위치 0은 지령이고 위치 1은 첫번째 인수, 위치 2은 두번째 인수 등이다. 실례로 내부지령 cd에 대하여 완성을 진행할 때 cd에 대한 첫번째(유일한) 인수는 그것이 등록부 이름일 때에만 완성되도록 한다고 하자. 이때 다음의 실례에서 보여 주는것처럼 완성을 프로그람화할수 있다.

complete cd 'p/1/d/'

complete지령뒤에 cd지령이 놓이고 그다음에 완성규칙이 놓인다. p는 지령행에서 단어위치를 표시한다. cd지령은 위치 0이고 그의 첫번째 인수는 위치 1이다. 규칙의 패턴부분이 빗선으로 둘러 싸여 있고(p/1/은 위치 1즉 cd에 대한 첫번째 인수를 의미한다) 완성규칙에 의해 영향을 받는다. 패턴의 d부분은 단어형태라고 한다. 단어형태들의 완성목록을 표 13-9에 주었다. d단어형태는 완성에 의해 등로부들만이 영향을 받는다는 것을 의미한다. 실례로 파일이름 또는 별명은 cd에 대한 첫번째 인수로 주어 질 때 완성되지 않는다. 규칙은 타브완성이 cd지령에서 진행될 때마다 첫번째 인수가 등록부일 때에만 진행되고 대조가 애매하면 Ctrl-D가 등록부를 표시만 하도록 한다. 이때 한개이상의 완성이 진행된다. p형완성을 실례 13-31에 주었다.

실례 13-31

 # p-type completions (positional completion)

1. > **complete**
   alias 'p/1/a/'
   cd 'p/1/d/'
   ftp 'p/1/(owl ftp.funet.fi prep.ai.mit.edu)'
   man 'p//c/'*

2. > **complete vi 'p/*/t/'**

3. > **complete vi**
   vi 'p//t/'*

4. > **set autolist**

5. > **man fill***[tab]* *# Completes command names*
   find find2perl findaffix findsmb finger

6. > **vi b***[tab]* *# Completes only filenames, not directories bashtest binded bindings bindit*

7. > **vi na***[tab]mes*

8. > **cd sh***[tab]ellsolutlons/*

9. > **set hosts = (netcom.com 192.100.1.10 192.0.0.200)**

10. > **complete telnet 'p/l/$hosts/'**

11. > **telnet net** *[tab] com.com*
    telnet netcom.corn

12. > **alias m***[tab]* *# Completes alias names*
    mc mroe mv

13. > **ftp prep***[tab]*

설명

1. complete내부지령은 인수가 없을 때 프로그람화된 모든 완성들을 표시한다. 다음의 실례들에서(행 2부터 11행까지) 이러한 완성들을 리용한다.
2. 이 규칙은 vi지령에 대한 인수를 입력할 때 타브완성을 리용하면 완성이 진행되도록 하기 위해 모든 인수(*)들을 t형(즉 보통본문파일)으로 한다.
3. complete지령은 지령이름을 인수로 리용할 때 그 지령에 대한 완성규칙을 현시한다. vi에 대한 완성규칙을 현시한다.
4. autolist내부지령을 설정하면 모든 가능한 타브완성들을 자동적으로 인쇄한다(ctrl-D를 누르지 말아야 한다.).

5. man지령은 프로그람화된 완성 즉 complete man '/p/1/c'를 가진다. 이 규칙은 man지령에 주어 진 첫번째 인수가 지령이여야 한다는것을 표시한다. 왜냐하면 c는 지령단어형태로 정의되여 있기때문이다. 이 실례에서는 문자 fin을 man에 대한 인수로 하여 완성을 하려고 한다. fin으로 시작되는 모든 조작지령들을 현시한다.
6. vi편집기완성은 본문파일만 완성하고 등록부들을 완성하지 않도록 프로그람화하였기때문에 파일이름들만 완성한다.
7. vi완성규칙에 따라서 얼마나 많은 인수들이 넘겨 지든 관계없이 본문파일이름들만 완성한다.
8. 파일이름완성이 내부 cd지령에 대한 첫번째 인수에서 진행될 때 완성되는 유일한 단어는 완성규칙에 의해서 서술된것과 같이 등록부이름이여야 한다. 이 실례에서 인수를 shellsolutions라고 하는 등록부로 확장한다.
9. 변수 hosts를 ID주소나 호스트이름들의 목록으로 설정한다.
10. telnet에 대한 완성규칙은 위치 1이 hosts변수에 설정된 호스트이름들중 어느 하나를 포함하는 완성이 진행되도록 한다. 이것은 목록단어형완성이다.
11. telnet지령을 실행하고 net로 시작하는 단어다음에 타브건을 누르면 그 단어가 netcom.com으로 완성되는데 이것은 앞에서 설정한 hosts변수에 있는 호스트이름들중의 하나이다.
12. 사용자가 단어 alias를, 그다음에 그 단어를 포함하는 모든 별명들로 확장되는 단어를 입력하면 alias완성을 진행한다. 단어형 a는 별명들만이 위치 1인 p를 위해 확장된다는것을 의미한다.

표 13-9. 완성단어형태

단 어	형 태
a	별명
b	편집기건맺기지령들
c	지령들(내부 또는 외부)
C	경로 앞붙이로 시작하는 외부지령들
d	등록부
D	경로 앞붙이로 시작하는 등록부들
e	환경변수들
f	파일이름들(등록부가 아니다.)
F	제공된 파일앞붙이로 시작하는 파일이름들
g	그룹이름들
j	일감들
l	한계

(표계속)

단 어	형 태
n	아무것도 아니다.
s	쉘변수들
S	신호들
t	보통(본문)파일들
T	제공된 경로앞붙이로 시작하는 보통(본문)파일들
v	임의의 변수들
u	사용자이름들
X	완성이 정의된 지령이름들
x	n과 같지만 ^D가 입력되면 통보문을 인쇄한다.
C,D,F,T	c, d, f, t와 같지만 주어 진 등록부로부터 완성들을 선택한다.
(list)	목록에 있는 단어들로부터 완성들을 선택한다.

c형완성을 리용하여 현재단어에 있는 패턴을 완성한다. 현재단어는 빗선안에 있는 패턴을 참조한다. 그것을 패턴과 대조하면 진행되는 임의의 완성이 패턴을 끌내도록 한다.

실례 13-32

 # c-type completions

1. > **complete**
   stty 'c/-/(raw xcase noflsh)/'
   bash 'c/-no/ (profile rc braceexpansion) / ‘
   find 'c/-/(user name type exec)/'
   man ‘c/perl/(delta faq toc data modlib locale)/’
2. > **stty -r***[tab]*aw
   stty -raw
3. > **bash -nop***[tab]rofile*
   bash -noprofile
4. > **find /** -n[tab]ame **.tcshrc -p***[tab]rint*
   find / -name .tcshrc -print
5. **> man perlde***[tab]lta*
   man perldelta
6. > **uncomplete stty**
   > **complete**
   bash 'c/-no/(profile rc braceexpansion)/'
   find *'c/-/(user* name type exec)/'
   man 'c/perl/(delta faq toc data modlib locale)/'
7. > **uncomplete ***

설명

1. 이 실례는 c형완성을 보여 준다. 빗선의 첫번째 모임에 있는 패턴을 입력하면 그 패턴은 그 목록으로부터 한개 문자를 입력하고 그다음에 타브건을 누를 때 괄호안에 있는 다른 목록들중의 어느 하나에 의해서 완성된다.
2. stty지령을 입력하고 그다음에 횡선(-)문자를 입력할 때 횡선, r 그리고 타브건을 입력하면 단어를 –raw로 완성한다. 괄호안에 있는 규칙목록(raw xcase noflsh)의 단어들중 한개의 단어를 완성할수 있다.
3. bash지령을 입력하고 그다음에 패턴 –no를 입력할 때 패턴 –no뒤에 p와 타브건이 놓이면 그 패턴을 –noprofile로 완성한다. 규칙목록(profile rc braceexpansion)에 있는 단어들중 어느 하나로부터 완성을 진행한다. 이 실례에서 –noprofile로 된다.
4. find규칙목록(user name type exec)에 있는 임의의 단어들중 중요한 문자들을 입력하여 횡선(-)문자를 완성하면 find지령에 대한 인수들을 완성한다.
5. man지령을 입력하면 패턴뒤에 목록(delta faq toc data modlib locale)의 한개 단어가 있으므로 패턴 perl을 perldelta로 완성한다.
6. uncomplete내부지령은 stty에 대한 완성규칙을 지우기한다. 다른 완성규칙들은 그대로 남아 있는다.
7. 별표식을 인수로 가지는 uncomplete내부지령은 완성규칙들을 지우기한다.

n형완성은 첫번째 단어를 대조하고 두번째 단어를 완성한다.

실례 13-33

 # n-type completions (next word completion)

1. > **complete**
   rm 'n/-r/d/ '
   find 'n/-exec/c/'
2. > **ls -ld testing**
   drwxr-sr-x 2 ellie root 1024 Aug 29 11:02 testing
3. > **rm -r te***[tab]sting*

설명

1. 이 실례는 n형완성을 보여 준다. 빗선의 첫번째 모임에 있는 단어(현재단어)를 입력하고 대조되면 다음(빗선의 두번째 모임에 있는)단어가 단어형태에 따라 완성된다. complate지령은 2개의 n형완성을 표시하는데 하나는 rm지령에 대한것이고 다른 하나는 find지령에 대한것이다. rm지령이 –r추가선택으로 실행될 때 –r다음에 있는 단어는 완성이 진행되려면 반드시 등록부형이여야 한다. find지령에 대한 규칙은 다음과 같다. 만일 –exec추가선택이 주어 질 때 완성이 진행되려면 그뒤에 놓이는 임의의 단어들은 지령이여야 한다.
2. ls지령의 출력은 testing이 등록부라는것을 보여 준다.
3. 파일이름완성은 단어완성을 testing이라고 하는 등록부에 대해서 진행하려고 하기기때문에 rm지령에 대하여 성공적으로 진행한다. testing이 보통파일이면 완성은 수행되지 않았을것이다.

4. 등록부탄창조작

만일 작업할 때 등록부나무를 많은 같은 등록부들에로 변경하려면 그것들을 등록부탄창에 넣고 탄창을 조작하여 그 등록부들에 쉽게 접근하게 할수 있다. 등록부탄창을 보통 식당에서 접시들을 쌓아 놓는것과 비교하는데 식당에서 접시들은 다른 접시의 우에 올라 가고 첫번째 들어 간 접시가 제일 밑바닥에 놓인다. pushd내부지령은 등록부들을 탄창에 넣고 popd지령들은 그것들을 지우기한다(다음의 실례들에서 보여 준다.). 탄창은 꼭대기의 등록부가 탄창에 제일 마지막으로 넣어 진 등록부로 되는 등록부들의 번호가 붙은 목록이다. 등록부들의 번호는 제일 꼭대기의 등록부가 0에서 시작하고 다음의것은 1로 되는 식으로 붙여 진다. dirs내부지령은 –v추가선택을 리용하여 번호가 붙은 등록부탄창을 현시한다.

pushd지령과 popd지령 등록부를 인수로 가지는 pushd지령은 새로운 등록부를 등록부탄창에 추가하며 동시에 그 등록부로 변화시킨다. 만일 인수가 횡선(-)이면 그것은 이전의 작업등록부를 표시한다. 인수가 +와 한개 수(n)로 이루어 지면 pushd는 탄창으로부터 n번째 등록부를 뽑아 내서 꼭대기에로 넣은 다음 그 등록부로 변경한다. 인수가 없으면 pushd는 등록부탄창의 꼭대기에 있는 2개의 요소들을 변경시켜 그것이 등록부들사이로 쉽게 이동할수 있게 한다. pushd가 작업하는 방식을 조종하는 많은 쉘변수들이 있다(813페지의 《국부변수설정》에서 보여 준다). 등록가입대화조종들이 바뀔 때마다 등록부탄창을 기억하기 위해 tcsh초기화파일들중 어느 하나(실례로 ~/.tcshrc)에 있는 savedirs변수를 설정해야 한다. 등록부탄창은 ~/.cshdirs라고 부르는 파일에 기억되고 쉘이 시동할 때 자동적으로 원천화된다. popd지령은 탄창의 꼭대기에서 한개 등록부를 지우기하고 그 등록부에로 변경시킨다.

표 13-10. 등록부탄창변수

pushdtohome	설정되면 인수가 없는 pushd는 pushd~ 또는 cd와 같다.
dunique	등록부를 탄창에 넣기전에 같은 이름을 가진 임의의 등록부를 지우기한다.
pushdsilent	Pushd를 실행할 때 등록부탄창을 인식하지 않는다.
deextract	설정되면 pushd+n은 n번째 등록부를 탄창에 넣기전에 등록부탄창으로부터 뽑아 낸다.
pushtohome	인수가 없으면 사용자의 홈등록부인 ~로 넣는다.
dirsfile	등록가입할 때마다 등록부탄창을 기억시킬수 있는 파일이름으로 대입한다.
savedirs	등록가입할 때마다 등록부탄창을 기억한다.
dirstack	등록부를 현시하거나 등록부를 그에 대입하는데 리용한다.

실례 13-34

1. > **pwd**
   /home/el1ie
   > **pushd ..**
   /home ~

> **pwd**
/home

2. > **pushd** *# Swap the two top directories on the stack*
~/home
> **pwd**
/home/ellie

3. > **pushd perlclass**

~/perlclass ~ /home

4. > **dirs -v**
0 ~/perlclass
1 ~
2 /home

Directory stack

0	*~/pelclass*
1	~
2	*/home*

5. >**popd**
~ */home*
> **pwd**
/home/ellie

6. >**popd**
/home/ellie

> **pwd**
/home

7. > **popd**
popd: Directory stack empty.

설명

1. pushd지령은 처음에 현재작업등록부인 /home/ellie를 표시한다. 다음에 pushd지령은 ..을 인수로 하여 어미등록부(..)를 등록부탄창에 넣는다. pushd의 출력은 /home이 등록부탄창의 꼭대기에 있으며 사용자의 홈등록부(~)인 /home/ellie는 탄창의 밑에 있다는것을 나타낸다. pushd는 또한 등록부를 탄창에 넣어 진 등록부 즉 /home으로 변경시킨다. 새로운 등록부는 두번째 pwd지령으로 표시된다.
2. 인수가 없는 pushd는 탄창에 있는 2개의 꼭대기등록부입력들을 교환하여 바뀌여 진 등록부로 변경시킨다. 이 실례에서 등록부는 사용자의 home등록부인 /home/ellie로 다시 절환된다.
3. pushd지령은 자기의 인수인 ~/perlclass를 탄창에 넣어 그 등록부로 변경시킨다.
4. 내부 dirs지령은 번호가 붙은 등록부탄창을 현시하는데 0은 제일 꼭대기등록부를 표시한다. 이 실례의 오른쪽에서 등록기탄창구조를 보여 준다.
5. popd지령은 탄창의 꼭대기에서 한개의 등록부를 지우기하여 그 등록부로 변경시킨다.
6. popd지령은 탄창의 꼭대기에서 또 다른 등록부를 지우기하여 그 등록부로 변경시킨다.

7. popd지령은 탄창이 비였기때문에 등록부입력을 더이상 지우기할수 없다. 따라서 오유통보를 내보내여 그것을 알려 준다.

5. 맞춤법교정

TC쉘에 추가된 특징인 맞춤법은 파일이름, 지령, 변수들에서 맞춤법오유를 정정하게 한다. 만일 emacs내부편집기를 리용하면 맞춤법오유는 Meta-s나 Meta-S건(Meta를 가지고 있지 않으면 Alt나 Esc건을 리용한다.)과 전체 행을 수정하는 Meta-$에 맺기된 맞춤법건들을 리용하여 정정할수 있다.[8] 재촉문의 값인 prompt 3은 맞춤법재촉문을 표시한다.

vi내부편집기를 리용하고 있다면 내부변수 correct를 설정하여 쉘이 맞춤법을 수정하도록 재촉한다.

실례 13-35

1. > **fimger** *[Alt-s]*   *# Replaces fimger with finger*
2. > **set correct=all**
3. **> dite**
   CORRECT>date (y/n/e/a)? yes
   Wed Aug 11 19:26:27 PDT 2001
   >
4. > **dite**
   CORRECT>date (y/n/e/a)? no
   dite: Command not found.
   >
5. > **dite**
   CORRECT>date (y/n/e/a)? edit
   >**dite**   *# Waits for user to edit and then executes command*
6. > **dite**
   CORRECT>date (y/n/e/a)? abort
   >

설명

1. meta(또는 Alt나 Esc)건을 s와 함께 누르면 지령, 파일이름, 변수의 맞춤법을 정정할수 있다. 만일 내부 vi편집기를 리용하고 있다면 이렇게 하지 못한다.
2. correct를 all로 설정하여 tcsh는 지령행에 있는 모든 맞춤법오유들을 정정하려고 한다. 이 특징을 emacs와 vi건맺기들에 대해서 리용할수 있다.
3. 지령이 틀렸기때문에 3차재촉문 prompt3인 CORRECT> date(y|n|e|a)?가

8. tcsh 사용지도서페지는 맞춤법교정을 사용자가 하려고 하는 방식을 수행하게 하는것이 아니라 실천적인 수단으로서 제공된다는 것을 알려 준다.

현시장치에 나타나고 사용자가 맞춤법을 요구하면 문자 y를 입력하고 요구하지 않으면 n을, 행을 직접 편집하려면 e를, 또는 전체 조작을 포기하려면 a를 입력하게 되여 있다.
4. 사용자가 지령을 변화시키지 않으려면 n을 입력한다.
5. 사용자가 편집상태에서 정정하려면 e를 누르고 그 지령을 직접 수정하거나 보강할수 있다.
6. 정정이 정확치 못하거나 요구되지 않으면 사용자는 a를 입력하여 맞춤법을 진행하지 않는다.

표 13-11. 정확한 변수인수

인 수	무엇을 하는가
cmd	지령들을 맞춤법으로 정정한다.
complete	지령들을 완성한다.
All	전체 지령행을 맞춤법으로 정정한다.

6. 별명

alias는 지령에 대한 TC쉘의 사용자정의된 생략표시이다.

별명들은 지령이 많은 추가선택이나 인수들을 가지거나 문법들을 기억하기 힘들 때 쓸모가 있다. 지령행에서 설정된 별명들은 자식쉘에 의해 계승되지 않는다. 별명들은 보통 .tcshrc파일에서 설정한다. .tcshrc은 새로운 쉘이 기동될 때 실행되기때문에 거기서 설정된 임의의 변경들은 새로운 쉘에 대해서 재설정된다. 별명들은 또한 쉘스크립트들에 넘겨 질수 있지만 그것들이 직접 스크립트안에서 설정되지 않으면 잠재적인 이식성문제들을 초래한다.

TC 쉘에는 몇가지 추가적으로 미리 설정된 별명들이 있는데 이것들은 사용자가 그것들을 정의할 때까지 정의하지 않은 상태로 남아 있다.

그러한것들로서는 beepcmd, cwdcmd, periodic, precomd 등 이다. 이 별명들은 842페지의 《특수한 별명》에서 표시되고 정의된다.

별명목록제시 alias내부지령은 모든 설정된 별명들의 목록을 현시한다. 별명이 먼저 인쇄되고 그다음에 실지 지령이나 그것이 나타내는 지령들을 인쇄한다.

실례 13-36

```
> alias
apache $HOME/apache/httpd –f $HOME/apache/conf/httpd.conf
co      compress
cp      cp –i
ls1     enscript –B –r –Porange –f Courier8 ! * &
mailq   /usr/lib/sendmail –bp
mc      setenv   MC '/usr/bin/mc –P !*'; cd $MC; unsetnv MC
mroe    more
mv      mv -i
uc      uncompress
```

uu uudecode
vg vgrind –t –s11 !:1 / lpr –t
weekly (cd/home/jody/ellie/activity; ./weekly_report; echo Done)

설명

alias지령은 첫번째 렬에 지령에 대한 별명을 표시하고 두번째 렬에 별명이 표시하는 실지 지령을 표시한다.

별명만들기 alias지령은 별명을 만드는데 리용한다. 첫번째 인수는 별명의 이름 즉 지령에 대한 가명이다. 행의 나머지부분은 별명이 실행될 때 실행되는 지령이나 지령들로 이루어 진다. 다중지령들은 반두점에 의해 분리되고 공백이나 메타문자들을 포함하는 지령들은 단일인용부호안에 놓인다.

형식

alias
alias 별명이름 지령
alias 별명이름 '지령 지령(들)'
unalias 별명이름

실례 13-37

1. > **alias m more**
2. > **alias mroe more**
3. > **alias lf ls-F**
4. > **alias cd 'cd \ !*; set prompt = "%/ •>**
5. > **cd ..**
6. /me/jody > ***cd /*** *# New prompt displyed*
   />
7. > *set tperiod = 60*
   > **alias periodic 'echo You have worked an hour, nonstop'**
8. > alias Usage 'echo "Error: \ i* " ; exit 1'

설명

1. more지령에 대한 별명을 m으로 설정한다.
2. more지령에 대한 별명을 mroe로 설정한다. 이것은 문자를 정확히 모를 때 편리하다.
3. 별명 lif는 tcsh내부지령 ls-F에 대한 가명이다. 그것은 ls-F와 같이 파일들을 표시하지만 더 빨리 한다.
4. cd가 실행할 때 cd에 대한 별명은 그것을 인수로 이름 지은 등록부로 가서 재촉문을 현재의 작업등록부 (%/)와 문자렬 >로 재설정하게 한다. !*는 리력기구에 의해 리용되는것과 같은 방법으로 별명으로 리용한다. 거꿀빗선은 리력기구가 별명이 !*를 리용하기전에 먼저 그것을 평가하지 못하도록 한다. \ !*는 리력목록에 있는 가장 최근의 지령으로부터 인수들을 표시한다.

공백때문에 별명정의를 인용부호안에 넣어야 한다.
5. cd지령을 등록부에로 변경시킨후 재촉문을 현재작업등록부(%/)와 >기호로 확장한다.ㄱ.
6. 새로운 등록부는 재촉문에 반영되는 /home/jody이다. 등록부를 뿌리(/)로 변화시킨후 재촉문이 다시 변화를 반영한다.
7. tperiod변수는 60분으로 설정된다. 별명 periodic는 미리 설정된 별명이다.
8. 이 별명은 통보를 내보내고 스크립트에서 탈퇴하기 위하여 스크립트에서 리용된다.

ㄱ. 사용자쉘로서 /bin/csh를 리용하려고 한다면 재촉문을 설정할 때 %/을 $cwd로 교체한다.

별명지우기 unalias지령을 별명을 지우는데 리용한다. 림시로 별명을 해제하기 위하여 별명이름앞에 거꿀빗선을 놓는다.

실례 13-38

1. > ***unalias mroe***
2. > \ ***cd..***

설명

1. unalias지령은 정의된 별명들의 목록으로부터 별명 more를 지운다.
2. 별명 cd는 이 지령이 실행될 때에만 림시로 해제된다.

별명순환 별명순환은 별명정의가 초기의 별명을 다시 참조하는 또 다른 별명을 참조할 때 생긴다.

실례 13-39

1. > **alias m more**
2. > **alias mroe m**
3. > **alias m more** *# Causes a loop*
4. > **m datafile**
   Alias loop.

설명

1. 별명은 m이다. 별명정의는 more이다. m을 리용될 때마다 more지령이 실행된다.
2. 별명은 mroe이다. 별명정의는 m이다. mroe를 입력하면 별명 m을 호출하고 more지령을 실행한다.
3. 이 지령은 아무것도 하지 않는 지령이다. 별명 m을 리용하면 그것은 별명 mroe를 호출하고 별명 mroe는 m을 다시 참조하여 별명순환을 일으킨다. 아무런 부정적인 효과도 일어 나지 않는다. 이때 다만 오유통보문이 표시된다.
4. 별명 m을 리용한다. 그것은 순환된다. m은 mroe를 호출하고 mroe는 m을, 그다음 m은 mroe를 호출하는 식으로 계속 순환을 진행한다. TC쉘은 무한순환을 진행하지 않고 그 문제점을 포착하여 오유통보문을 현시한다.

제 4 절. 일감조종

일감조종은 TC쉘의 강력한 수단으로 배경이나 전경에서 jobs프로그람을 실행할수 있게 한다. 일반적으로 지령행에서 입력된 어떤 지령은 전경에서 실행하며 끝날 때까지 계속된다. 만일 창문에서 프로그람을 작성한다면 새 일감을 시작하기 위해 다만 다른 창문을 열수 있기때문에 일감조종이 필요하지 않을수도 있다. 한편 단일말단에서 일감조종은 매우 쓸모 있는 기능이다.

일감지령의 목록을 표 13-12에 주었다.

표 13-12. 일감조종지령

지 령	의 미
jobs	실행되는 모든 일감들을 표시한다.
^Z(Ctrl-Z)	일감을 정지(중지)한다. 재촉문이 현시장치에 나타난다.
bg	배경에서 정지된 일감의 실행을 시작한다.
fg	배경일감을 전경으로 가져 간다.
kill	지적된 일감에 kill신호를 보낸다.

일감지령인수	의 미
%n	일감번호 n
%string	문자렬로 시작하는 일감이름
%?stribg	문자렬을 포함하는 일감이름
%%	현재일감
%+	현재일감
%-	현재일감이전의 일감

1. 배경일감

앰퍼샌드 어떤 지령을 완성하는데 오랜 시간이 걸린다면 &를 지령에 추가하여 그 일감이 배경에서 실행되게 할수 있다. tcsh재촉문은 즉시에 귀환하며 이때 다른 지령을 입력할수 있다. 2개의 지령을 동시에 실행하는데 하나는 전경에서, 다른 하나는 배경에서 실행한다. 이것들은 둘다 자기의 표준출력을 현시장치에 보낸다. 배경에 어떤 일감을 배치하면 그 출력을 파일에 방향바꾸기하거나 인쇄기와 같은 장치에 파이프하는것이 좋다.

실례 13-40

```
1. > find . –name core –exec rm {} \ ; &
2. [1]  543
3. >
```

설명

1. find지령을 배경에서 실행한다(-print추가선택이 없는 find지령은 어떤 출력도 현시장치에 보내지 않는다.).ㄱ
2. 중괄호에 있는 수자는 이것이 배경에서 실행되는 첫 일감이라는것을 보여주며 이 프로쎄스의 PID는 543이다.
3. 재촉문이 즉시에 귀환한다. 쉘은 사용자입력을 기다린다.

ㄱ. find문법은 exec명령문의 끝에 반두점을 요구한다. 반두점이 \ 앞에 놓여 쉘이 그것을 해석하지 않도록 한다.

중단건문자 프로그람을 중단시키기 위하여 중단건문자렬 ^Z를 발생시킨다. 이때 일감은 중단(정지)되고 쉘재촉문이 현시되며 프로그람은 fg 또는 bg지령이 발생할 때까지 다시 시작하지 않는다(vi편집기를 리용할 때 ZZ지령은 파일을 쓰고 기억한다. 이것을 vi 대화조종을 중단시키는 ^Z와 혼돈하지 말아야 한다.). 일감이 중단될 때 등록탈퇴하려고 하면 통보문《중단된 일감이 있다.》가 현시장치에 나타난다.

jobs지령과 listjobs변수 tcsh내부지령 jobs는 현재 진행된는 프로그람과 배경에서 실행되거나 중단된 프로그람들을 현시한다. 실행한다는것은 일감이 배경에서 실행되고 있다는것을 의미한다. 일감이 중단될 때 그것은 정지되며 실행상태에 있지 않다. 두 경우에 말단은 해방되여 다른 지령을 접수한다. 일감이 정지된 동안 쉘을 탈퇴하려고 시도하면 경고 《중단된 일감들이 있다.》가 현시장치에 현시된다. 즉시에 다시 탈퇴하려고 시도하면 쉘은 계속하여 중단된 일감들을 완료한다. 일감을 중단시킬 때 통보문을 자동적으로 인쇄하려고 한다면 tcsh의 내부변수 listjobs를 설정한다.

실례 13-41

(The Command Line)

1. > **jobs**
2. *[1] + Suspended vi filex*
   [2] - Running sleep 25
3. > **jobs –1**
   [1] +355 Suspended vi filex
   [2] -356 Running sleep 25
4. *[2] Done sleep 25*
5. > **set listjobs = long**
   > **sleep 1000**
   Press Control-Z to suspend job

   [1] + 3337 Suspended sleep 1000
   >
6. > **set notify**

설명

1. jobs지령은 현재능동일감들을 표시한다.
2. 표기 [1]은 첫 일감의 번호인데 +기호는 일감이 배경에 배치된 가장 최근의 일감이 아니라는것을 나타낸다. 횡선은 이것이 배경에 넣은 가장 최근의 일감이라는것을 나타내며 중단되였다는것은 이 일감이 ^Z에 의해 정지되며 현재 비능동상태라는것을 의미한다.
3. –l추가선택(긴 표시)은 일감의 PID는 물론이고 일감의 번호를 표시한다. 표기[2]는 두번째 일감의 번호이다. 이 경우에는 배경에 배치된 마지막 일감이다. 횡선은 이것이 가장 최근의 일감이라는것을 표시한다. sleep지령은 배경에서 실행되고 있다.
4. sleep가 25s동안 실행된후에 일감은 완성되고 그것이 끝났다고 알려 주는 통보문이 현시장치에 나타난다.
5. tcsh 의 listjobs변수는 long으로 설정하였는데 중단될 때 그의 프로쎄스 ID 번호는 물론 일감의 번호도 인쇄한다(표 13-25에서 내부 tcsh변수들의 목록을 주었다.).
6. 일반적으로 쉘은 재촉문을 인쇄하기전에 일감이 끝나면 그것을 알려 주지만 쉘변수 notify가 설정되면 쉘은 배경일감의 상태에 어떤 변화가 생길 때 즉시에 알려 준다. 실례로 vi편집기로 작업하고 있고 배경일감이 완료되면 통보문이 vi창문에 다음과 같이 즉시에 나타난다.
   [1]   Terminated   sleep20

2. 전경과 배경지령

fg지령은 배경일감을 전경에 가져 온다. bg지령은 배경에서 실행되고 있는 중단된 일감을 시작한다. 일감조종을 위한 특별한 일감을 선택하려고 한다면 %기호와 일감번호를 fg와 bg에 대한 인수로서 리용할수 있다.

실례 13-42

1. > **jobs**
2. *[1] + Suspended   vi filex*
3. *[2] – Running   cc prog.c –o prog*
   > **fg %1**
   vi filex
   (vi   session   starts)
4. > **kill   %2**
   [2]   terminateed   c   prog.c   -o   prog
5. > **sleep 15**
   (Press ^z)
   Suspended
6. > **bg**
   [1] sleep   15 &
   done   sleep 15

설명

1. jobs지령은 일감이라고 부르는 현재 실행되고 있는 프로세스를 표시한다.
2. 첫 일감은 vi대화조종(중단된)이며 두번째는 cc지령(실행되는)이다.
3. [1]이라고 번호를 단 일감을 전경에로 보낸다. 번호앞에 퍼센트기호가 놓인다.
4. kill지령은 내부지령이다. 그것은 TREM(완료)신호를 기정으로 프로세스에 보낸다. 인수는 프로세스의 번호 혹은 PID이다.
5. sleep지령은 ^Z를 누르면 정지된다. sleep지령은 CPU를 쓰지 않고 배경에서 중지된다.
6. bg지령은 마지막배경일감이 배경에서 실행을 시작하도록 한다. sleep프로그람은 실행이 다시 시작하기전에 초로 내리계수를 시작한다.ㄱ

ㄱ. grep, sed, awk 와 같은 프로그람들은 패턴종합을 위하여 정규표현메타문자라고 부르는 메타문자들의 모임을 가지고 있다. 이것들을 쉘메타문자들과 혼돈하지 말아야 한다.

3. 일감일정작성

sched내부지령은 지정된 시간에 실행되도록 순서를 작성한 일감목록을 만들수 있게 한다. 인수가 없는 sched지령은 모든 순서짜기사건들의 번호 붙은 목록을 표시한다. 그것은 시간을 hh:mm형식으로 설정하는데 여기서 시간은 군사용이거나 12시간 AM/PM 형식일수 있다. 시간은 또한 +기호를 가진 상대시간 즉 현재시간에 대한 상대시간으로 지정할수도 있으며 –기호를 리용하면 사건이 목록에서 지우기된다.[9]

형식

```
sched
sched [+]hh:mm   지령
 sched –n
```

실례 13-43

1. **> sched 14:30   echo '*G Time to start your lecture!'**
2. **> sched 5PM   echo Time to go home.**
3. **> sched +1:30 /home/ellie/scripts/logfile.sc**
4. **> sched**
   1   17:47 /home/scripts/logflie.sc
   2   5PM   echo Time to go home.
   3   echo '^G Time to start your lecture!'
5. **> sched -2**
   > sched
   1 17:47 /home/scripts/logfHe.sc
   2 14:30   echo '^G Time to start yoW lecture! '

---

9. tcsh의 사용지도서페지는 순서짜기사건목록에 있는 지령이 그것이 계획된후에 첫번째 재촉문이 인쇄되기직전에 실행된다는것을 알려 준다. 만일 지령이 실행되는 정확한 시간이 생략되면 overdue 지령을 다음재촉문에서 실행한다.

설명

1. sched지령은 echo지령이 14시 30분에 실행되도록 일정을 작성한다. 이때 경고음이 나고(Control-G)ㄱ 통보문이 표시된다.
2. sched지령은 echo지령이 오후 5시에 실행되도록 일정을 작성한다.
3. 스크립트 logfile.sc는 지금부터 1시간 30분이 지나서 실행되도록 일정을 작성한다.
4. sched지령은 크기순서로 일정이 작성된 사건들을 현시한다. 즉 마지막사건이 먼저 현시된다.
5. 수값인수를 가진 sched는 일정을 작성한 목록으로부터 번호 붙은 일감을 지우기한다. 일감번호 2가 sched의 출력에서 보여 준것처럼 지우기된다.

ㄱ. ^G를 echo명령문에로 보내기 위하여 Ctrl-M을 뒤따르는Ctrl-V를 입력하고 그 다음에 Ctrl-G를 입력한다.

제 5 절. 메타문자

메타문자는 그 자체가 아닌 다른것을 표현하는데 리용된 특수한 문자이다. 일반적으로 보면 문자나 수자가 아닌 문자가 메타문자일수 있다. 쉘은 흔히 쉘통용기호라고 하는 자기의 메타문자들을 가지고 있다. 쉘메타문자들을 리용하여 지령들을 묶고 파일이름과 경로이름을 간략화하며 입출력을 방향바꾸기하고 파이프하여 지령들을 배경에 놓는것과 같은것들을 할수 있다. 표 13-13에서 쉘메타문자들의 부분목록을 보여 준다.

표 13-13. 쉘메타문자

메타문자	목 적	실 례	의 미
$	변수바꿔넣기	set name=Tom echo $name Tom	변수 name을 Tom으로 설정한다. 거기에 기억된 값을 현시한다.
!	리력바꿔넣기	!3	리력목록으로부터 세번째 사건을 재실행시킨다.
*	파일이름바꿔넣기	rm*	모든 파일들을 지우기한다.
?	파일이름바꿔넣기	ls??	두 문자로 된 모든 파일들을 표시한다.
[ ]	파일이름바꿔넣기	catf[123]	f1, f2, f3의 내용들을 현시한다.
;	지령분리기호	ls;date;pwd	매 지령이 차례로 실행된다.
&	배경처리	lp mbox&	배경에서 인쇄가 진행된다. 재촉문이 즉시에 복귀한다.
>	출력방향바꾸기	ls>file	표준출력을 file에로 방향바꾸기한다.
<	입력방향바꾸기	ls<file	표준입력을 file로부터 방향바꾸기한다.
>&	출력과 오유의 방향바꾸기	ls>&file	출력과 오유를 둘다 file에로 방향바꾸기 한다.

(표계속)

메타문자	목 적	실 례	의 미
>!	noclobber가 설정되면 무시한다.	ls>!file	파일이 존재하면 noclobber가 설정되였어도 그 끝을 자르고 겹쳐쓰기한다.
>>!	noclobber가설정되면 무시한다.	ls>>!file	파일이 존재하지 않으면 noclobber가 설정되여도 그것을 작성한다.
( )	지령들을 묶어서 자식쉘에서 실행되게 한다.	(lp;pwd)>tmp	지령들을 실행시켜 출력을 tmp파일에로 보낸다.
{ }	지령들을 묶어서 이 쉘에서 실행시킨다.	{cd/;echo $cwd}	뿌리등록부로 등록부를 변화시키고 현재작업등록부를 현시한다.

1. 파일이름바꿔넣기

지령행을 평가할 때 쉘은 어떤 문자모임과 대조하는 파일이름이나 경로이름들을 간략화하기 위해 메타문자들을 리용한다. 표 13-14에서 보여 준 파일이름바꿔넣기메타문자들은 자모문자로 표시된 파일이름들의 모임으로 확장된다. 메타문자를 파일이름으로 확장하는 처리를 메타문자파일이름확장이라고 부른다. 다른 쉘들과 달리 C쉘이 파일이름으로 그것을 표시하게 된 메타문자들을 교체할수 없을 때 No match를 통보한다.

표 13-14. 쉘메타문자와 파일이름바꿔넣기

메타문자	의 미
*	여러개의 물음표들을 대신한다.
?	한개 문자만 대신한다.
[abc]	a, b, c문자모임에서 한개 문자를 대신한다.
[a-z]	a와 z사이에 있는 한개 문자를 대신한다.
[^abc]	a, b, c가 아닌 임의의 물음표를 대신한다.
[a,ile,ax]	한개 문자나 문자들의 모임을 대신한다.
~	사용자의 홈등록부로 물결표를 교체한다.
\	메타문자를 탈퇴시키거나 불가능하게 한다.

메타문자확장 쉘은 자기의 메타문자들을 평가하고 그것들을 파일이름에서 적당한 문자나 수자로 교체하여 파일이름바꿔넣기를 진행한다.

별표 별표는 파일이름에서 여러개의 물음표들을 대신한다.

실례 13-44

```
1.  > ls
    a.c h.c abc ab3 filel file2 file3 file4 file5
```

2. > **echo ***
   a.c b.c abc ab3 filel file2 file3 file4 file5

3. > **ls   *.c**
   a.c b.c

4. > **ls ^*.c**
   abc ab3 filel file2 file3 file4 file5

5. > **rm   z*p**
   No match.

설명

1. 현재등록부에 있는 모든 파일들을 표시한다.
2. echo프로그람은 모든 인수들을 현시장치에 인쇄한다. 별표는 match for zero or more of any characters found in a filename을 의미하는 통용기호이다. 등록부에 있는 모든 파일들이 대조되여 현시장치에 출력된다.
3. .c로 끝나는 파일이름이 표시된다.
4. .c로 끝나지 않는 모든 파일들이 표시된다. 등록부에 있는 어느 파일도 z로 시작하지 않으므로 쉘은 No match라고 통보한다.

물음표 물음표는 파일이름에서 한개의 문자만을 대신한다.

실례 13-45

1. > **ls**
   a.c b.c abc ab3 filel £ile2 file3 file4 file5

2. > **ls ???**
   abc ab3

3. > **echo How are you\ ?**
   No match.

4. > **echo How are you\ ?**
   How are you?

설명

1. 현재등록부에 있는 모든 파일들을 표시한다.
2. 물음표는 한 문자파일이름에 대조된다. 3문자로 이루어 진 모든 파일들이 표시된다.
3. 쉘은 한개의 문자가 뒤에 붙은 y-o-u라는 파일이름을 찾는다. 등록부에 3문자로 된 파일이 없다. 쉘은 No match라고 통보한다.
4. 물음표앞에 거꿀빗선을 리용하여 물음표의 특수한 의미를 없앤다. 이때 쉘은 물음표를 문자그대로 처리한다.

중괄호 중괄호([])는 파일이름에서 문자모임이나 령역에 있는 한개 문자를 대신한다.

실례 13-46

1. > **ls**
   a.c b.c abc ab3 filel file2 file3 file4 file5 file10
   filell filel2
2. > **ls file[123]**
   filel file2 file3
3. > **ls file[^123]**
   file4 file5
4. > **ls [A-Za-zl [a-z] [1-5]**
   ab3
5. > **ls file1[0-2]**
   file10 file11 file12

설명

1. 현재등록부에 있는 모든 파일들이 표시된다.
2. file로 시작하고 그뒤에 1, 2 또는 3이 오는 파일이름들이 대조되여 표시된다.
3. file로 시작되고 그뒤에 1, 2, 3이 아닌 단일문자가 오는 파일이름들이 대조되여 표시된다.
4. 한개 문자(소문자나 대문자)로 시작되고 그뒤에 한개 소문자와 1과 5사이의 수자가 오는 파일이름들이 대조되여 표시된다.
5. file1로 시작하고 그다음에 0, 1 또는 2가 놓이는 파일이름들이 표시된다.

대괄호 대괄호({})는 이미 존재할수도 있고 없을수도 있는 파일이름의 한개 문자나 문자렬들을 대신한다.

실례 13-47

1. > **ls**
   a.c b.c abc ab3 ab4 ab5 filel file2 file3 file4 file5 foo faa fumble
2. > **ls f{oo,aa,umble}**
   foo faa fumble
3. > ls a{.c,c,b[3-5]}
   a.c ab3 ab4 ab5
4. > mkdir prog{l,2,3}
5. >**echo tweedle{dee,dum}, {1,cl,m}ove{r}**
   tweedledee tweedledum, lover clover mover

설명

1. 현재등록부에 있는 모든 파일들이 표시된다.
2. f로 시작하고 문자렬 oo, aa 또는 umble이 뒤따르는 파일들이 표시된다. 대괄호안에 있는 공백들은 오유통보문 Missing을 발생한다.
3. a로 시작하고 그뒤에 .c, c나 b3, b4 혹은 b5가 오는 파일들이 표시된다(중괄호들은 대괄호안에서 리용할수 있다.).

4. prog1, prog2, prog3과 mkdir에 대한 개별적인 인수들을 입력하지 않고 대괄호를 리용하여 새 등록부이름을 만들수 있다.
5. 괄호안에 있는 부분을 앞붙이나 뒤붙이로 리용하여 매 단어들을 확장한다. 여기서 중요한것은 대괄호안에 있는 단어들이 공백을 포함하지 않는것이다.

메타문자의 탈퇴와 nonomatch변수 거꿀빗선은 한개 문자의 특수한 의미를 없애는데 리용된다. 특수한 의미에서 벗어 난 문자는 그자체를 표시한다.

실례 13-48

1. > **got milk?**
   got: No match.
2. > **got milk\ ?**
   got: Command not found.
3. > **set nonomatch**
   > **got milk?**
   got: Command not found.

설명

1. 물음표는 파일바꿔넣기메타문자로서 한개 문자와 같다. 쉘은 작업등록부에서 한개 문자가 뒤에 놓이는 m-i-l-k를 포함하는 파일을 찾는다. 쉘이 그 파일을 찾지 못하면 No match를 통보한다. 이것은 쉘이 지령행을 구문해석하는 순서를 보여 준다. 쉘이 got지령을 찾기전에 메타문자들을 평가한다.
2. 거꿀빗선은 메타문자가 해석되지 않게 하는데 이것을 보통 메타문자에서 벗어나기라고 한다. 이때 쉘은 No match를 통보하지 않지만 got지령에 대한 경로를 탐색하는데 그것을 찾지 못한다.
3. 내부 nonomatch변수는 설정될 때 메타문자가 대조되지 않으면 오유통보문을 내보내지 않는다. No match를 통보하지 않고 UNIX는 got가 지령이 아니라고 통보한다.

물결표확장 물결표기호(～)는 단독으로 사용자홈등록부의 완전경로이름으로 확장된다. 물결표가 사용자이름앞에 붙으면 그 사용자의 홈등록부의 완전경로이름으로 확장된다. 경로이름앞에 붙으면 홈등록부와 경로이름의 나머지부분으로 확장된다.

실례 13-49

1. > **echo ~**
   /home/jody/ellie
2. > **cd ~/desktop/perlstuff**
   > **pwd**
   /home/jody/ellie/destop/perlstuff
3. > **cd ~joe**
   > **pwd**
   /home/bambi/joe

설명

1. 물결표를 사용자의 홈등록부에로 확장한다.
2. 물결표뒤에 경로이름이 놓이면 /desktop/perlstuff가 뒤에 놓이는 사용자의 홈등록부로 확장된다.
3. 사용자이름이 뒤에 놓이는 물결표는 사용자의 홈등록부에로 확장된다. 이 실례에서 등록부는 그 사용자의 홈등록부로 변경된다.

Noglob에 의한 메타문자해제 noglob변수를 설정하면 파일이름바꿔넣기는 해제되고 모든 메타문자들이 자기자체를 표시한다. 즉 그것들을은 wildcard로서 리용되지 않는다. 이것은 grep나 sed, awk와 같은 프로그람에서 패턴을 찾을 때 효과가 있는데 그 프로그람들은 쉘이 확장하려고 하는 메타문자들을 포함할수도 있다.

실례 13-50

```
1. > set  noglob
2. > echo * ??  [] ~
   * ?? [] ~
```

설명

1. 변수 noglob가 설정된다. 그것은 wildcard들의 특수의미를 해제한다.
2. 메타문자들을 어떤 해석도 없이 그 자체로 현시한다.

제 6 절. 방향바꾸기와 파이프

일반적으로 지령으로부터 표준출력 (stdout) 은 현시장치로 보내고 표준입력 (stdin) 은 건반으로 들어 오며 오유통보문(stden)은 현시장치에로 간다. 쉘은 파일에로 또는 파일로부터 입력와 출력을 방향바꾸기하기 위해 특수한 방향바꾸기메타문자들을 리용하게 한다. 방향바꾸기연산자들(＜, ＞, ＞＞, ＞&)뒤에 파일이름이 놓인다. 이 파일은 왼쪽에 있는 지령이 실행되기전에 쉘에 의해 열린다. 수직선(｜)기호에 의해 표시되는 파이프는 한 지령의 출력을 다른 지령의 입력으로 보낸다. 파이프의 왼쪽에 있는 지령은 그것을 파이프에 쓰기때문에 작성자라고 부른다. 오른쪽 지령은 파이프로부터 읽기때문에 독자라고 부른다. 방향바꾸기와 파이프메타문자들의 목록을 표 13-15에 주었다.

표 13-15. 방향바꾸기메타문자

메타문자	의 미
command < file	파일로부터 지령에로 입력을 방향바꾸기한다.
command > file	지령으로부터 파일로 출력을 방향바꾸기한다.
command >&file	출력과 오유를 파일에 방향바꾸기한다.
command >>file	지령의 출력을 방향바꾸기하여 파일에 추가한다.

(표계속)

메타문자	의 미
command >>&file	지령의 출력과 오유를 방향바꾸기하여 파일에 추가한다.
command << WORD	첫번째 WORD로부터 마지막 WORD까지의 입력을 지령에로 방향바꾸기한다.
<input>	사용자입력을 여기서 진행한다. 2중인용부호가 붙은 본문문자렬로서 취급한다.
WORD	WORD는 지령에 대한 입력의 끝을 표시한다.
command |command	첫번째 지령의 출력을 두번째 지령의 입력에로 파이프한다.
command |&command	첫번째 지령의 출력과 오유를 두번째 지령의 입력에로 파이프한다.
command>!file	noclobber변수를 설정하면 이 지령에 대한 영향을 무시하고 파일을 열거나 그우에 겹쳐쓰기한다.
command >>! File	noclobber변수를 무시한다. 파일이 존재하지 않으면 파일을 작성하고 지령으로부터의 출력을 그 파일에 추가한다.
command>>&!file	noclobber변수를 무시한다. 파일이 존재하지 않으면 그것을 작성하고 출력과 오유를 그 파일에 추가한다.

1. 입력방향바꾸기

입력을 말단건반이 아니라 파일로부터 방향바꾸기할수 있다. 쉘은 <기호의 오른쪽에 있는 파일을 열고 왼쪽에 있는 프로그람을 그 파일로부터 읽어 들인다. 파일이 존재하지 않으면 오유 No such file or directory가 쉘에 의해 통보된다.

형식

command < file

실례 13-51

mail bob < memo

설명

파일 memo가 쉘에 의해 열리고 입력을 mail프로그람에로 방향바꾸기한다. mail프로그람에 의해 사용자 bob에게 memo라고 부르는 파일을 보낸다.

2. here문서

here문서는 인용부호안에 있는 본문블로크형식의 지령에로 입력을 방향바꾸기하는 또 다른 방법으로서 차림표를 만들고 다른 프로그람으로부터 입력을 처리하기 위하여 쉘

스크립트에서 사용한다. 일반적으로 건반으로부터 입력을 접수하는 프로그람들은 Ctrl-D(^D)로 완료된다. here문서는 ^D를 입력하지 않고 입력을 프로그람에 보내고 완료하는 다른 방법을 제공한다. <<기호뒤에는 중단기호(terminator)라고 부르는 사용자정의된 단어가 놓인다. 입력은 사용자정의된 완료자가 나타날 때까지 <<기호의 왼쪽에 있는 지령에로 방향바꾸기된다. 최종완료자는 단독으로 행에 놓이며 그 주위에 어떤 공백도 놓일수 없다. 변수와 지령바꿔넣기는 here문서안에서 수행된다. 보통 here문서는 mail, bc, ex, ftp 등과 같은 지령들에 입력을 제공하고 차림표를 작성하기 위하여 쉘스크립트에서 사용한다.

형식

```
Command << MARK
        ... input...
MARK
```

실례 13-52

(Without the here document)

(The Command Line)

1. > **cat**
2. Hello There.
   How are you?
   I'm tired of this.
3. ^d

(The Output)

4. *Hello There.*
   How are you?
   I'm tired of this.

설명

1. cat프로그람은 인수가 없으면 건반입력을 기다린다.
2. 사용자는 건반에서 입력한다.
3. 사용자는 입력을 끝내기 위하여 cat프로그람에 ^D를 입력한다.
4. cat프로그람은 그의 출력을 현시장치에 보낸다.

실례 13-53

(With the *here document*)

(The Command Line)

1. > **cat << DONE**

2. ? Hello There.
   ? How are you?
   ? I'm tired of this.
3. ? **DONE**
4. *Hello There.*   *<----The output from the here document*

   How are you?

   *I'm tired of this.*cat

설명

1. 프로그람은 첫번째 DONE으로부터 마지막 DONE까지 입력을 받는다. 단어들은 사용자정의된 완료자이다.
2. 물음표(?)는 2차재촉문 prompt2이다. 그것은 here문서가 사용자정의된 완료자 DONE으로 끝날 때까지 나타난다. 이 행은 입력이다. 단어 DONE이 나타날 때 더이상 입력을 받지 않는다.
3. 최종완료자는 입력의 끝을 표시한다. 이 단어의 앞뒤에는 공백이 있을수 없다.
4. 첫 단어 DONE과 마지막단어 DONE사이의 본문은 cat지령의 출력으로서 현시장치에 보낸다. 마지막단어 DONE은 그것의 오른쪽에 공백이나 다른 본문이 없이 왼쪽 경계에 있어야 한다.

실례 13-54

(The Command Line)

1. > set name = steve
2. > **mail $name << EOF**
3. ? Hello there, $name
4. ? The hour is now 'date +%H'
5. ? **EOF**
6. >

설명

1. 쉘변수 name은 사용자이름 steve로 대입된다(보통 이 실례는 쉘스크립트에 포함된다.).
2. 변수 name은 here문서안에서 확장된다.
3. 물음표(?)는 2차재촉문 prompt2이다. 그것은 here문서가 사용자정의된 완료자 EOF로 끝날 때까지 나타난다. mail프로그람은 완료자 EOF가 나타날 때까지 입력을 받는다.
4. 지령바꿔넣기는 here문서안에서 수행된다. 즉 거꿀인용부호안에 있는 지령이 실행되고 그 출력이 문자렬안에서 교체된다.
5. 완료자 EOF가 나타나면 mail프로그람에 대한 입력을 정지한다.

3. 출력방향바꾸기

기정으로 한개 지령이나 지령들의 표준출력은 말단현시장치로 나간다. 현시장치로부터 파일에로 표준출력을 방향바꾸기하기 위하여 >기호를 쓴다. 지령은 >기호의 왼쪽에 있고 파일이름은 오른쪽에 있다. 쉘은 >기호의 오른쪽에 있는 파일을 연다. 파일이 존재하지 않으면 쉘은 그것을 작성하고 존재한다면 파일을 열고 그것을 자른다. 보통 방향바꾸기를 리용할 때 부주의로 파일들이 지워 진다(noclobber라고 하는 tcsh특수변수는 방향바꾸기할 때 현존 파일을 지우지 않도록 하기 위하여 설정될수 있다. 표 13-16에서 보여 준다.).

형식

command > file

실례 13-55

cat file1 file2 > file3

설명

file1와 file2의 내용들은 련결되여 그 출력이 file3에 보내진다. 쉘이 cat지령을 실행하려고 하기전에 file3을 연다는것을 알아야 한다. file3이 존재하지 않으면 작성된다.

출력을 현존파일에로 추가 출력을 현존파일에로 추가하기 위하여 >>기호를 리용한다. >>기호의 오른쪽에 있는 파일이 존재하지 않으면 만들어 지고 존재한다면 그 파일이 열리고 출력이 그의 끝에 추가된다.

형식

command >> file

실례 13-56

data >> outfile

설명

data지령의 표준출력은 outfile에 방향바꾸기되여 추가된다.

출력과 오유를 방향바꾸기 >&기호는 표준출력과 표준오유를 둘다 파일에로 방향바꾸기 하는데 리용된다. 일반적으로 어떤 지령이 성공하여 출력을 stdout에 보내든지 실패하여 오유통보문을 stderr에 보낸다. find와 du와 같은 일부 재귀프로그람들은 그것들이 등록부나무를 거쳐 이동할 때 표준출력과 오유를 현시장치에 보낸다. >&기호를 리용할 때 표준출력과 표준오유는 둘다 파일에 기억되여 실행될수 있다. TC쉘은 오직 표준오유만을 위한 방향바꾸기기호를 제공하지 않지만 자식쉘에서 지령을 실행하여 표준오유를 얻을수 있다. 그림 13-3에서 보여 준다.

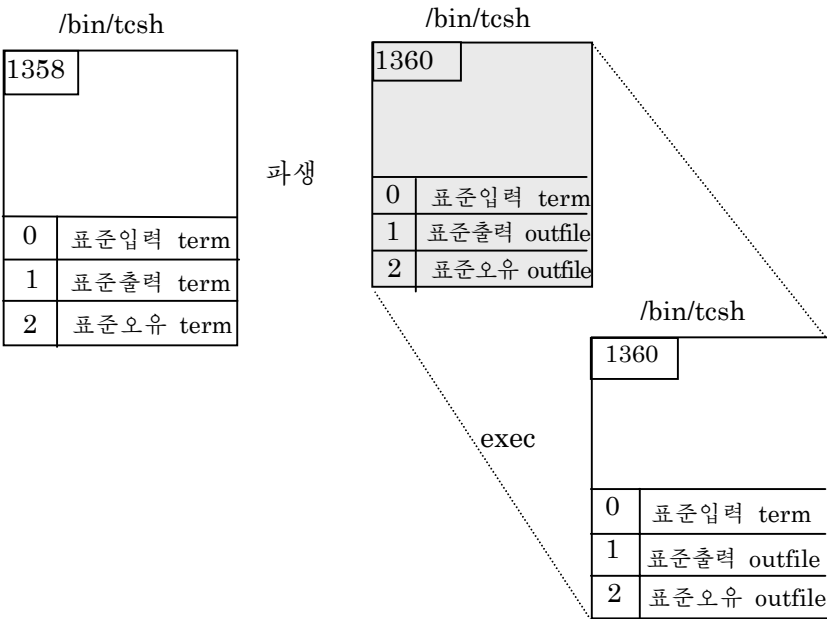



그림 13-3. 표준출력과 표준오유의 방향바꾸기

실례 13-57

1. > **date**
   Tue Aug 3 10:31:56 PDT 2001
2. > **date >& outfile**
3. > **cat outfile**
   Tue Aug 3 10:31:56 PDT 2001

설명

1. date지령의 출력을 표준출력인 현시장치에 보낸다.
2. 출력과 오유를 outfile에 보낸다.
3. 오유가 없기때문에 표준출력을 outfile에 보내고 파일의 내용을 현시한다.

실례 13-58

1. > **cp file1 file2**
2. > **cp file1**
   cp: missing destination
   Try 'cp –help' for more information
3. > **cp file1 >& errorfile**
4. **> cat errfile**
   cp: missing destination file
   Try 'cp –help' for more information

설명

1. Try 'cp –help' for more information파일을 복사하기 위하여 cp지령은 원천파일과 목적파일을 둘다 요구한다. cp지령은 file1의 사본을 만들고 그것을 file2에 넣는다. cp지령을 정확한 문법으로 주면 현시장치에 아무것도 현시되지 않는다.
2. 이때 목적파일이 없으므로 cp지령이 실패하고 오유를 말단인 stderr에 보낸다.
3. >&기호는 stdout와 stderr를 둘다 errorfile에 보내는데 리용된다. 지령으로부터 출력만이 오유통보문이기때문에 그것이 errorfile에 기억된다.
4. errorfile의 내용을 현시하여 그것이 cp지령에 의해 생기는 오유통보문을 포함한다는것을 보여 준다.

출력과 오유분리 표준출력과 표준오유는 지령들을 괄호안에 넣어 분리할수 있다. 지령이 괄호안에 있을 때 T C쉘은 자식쉘을 시작하고 자식쉘안에서부터 방향바꾸기를 처리한 다음에 지령을 실행한다. 실례 13-59에서 보여 준 수법을 리용하여 표준출력을 오유로부터 분리할수 있다.

실례 13-59

(The Command Line)

1. **> find . –name '*.c' >& outputfile**
2. **> (find . –name '*.c'> goodstuff) >& badstuff**

설명

1. find지령은 현재등록부에서 시작하여 .c로 끝나는 모든 파일들을 찾고 outputfile에 그 출력을 인쇄한다. 오유가 생기면 그것도 outputfile에로 보낸다.
2. find지령은 괄호안에 포함된다. 쉘은 지령을 처리하기 위해 자식쉘을 만든다. 자식쉘을 만들기전에 괄호밖의 단어들을 처리한다. 즉 badstuff파일이 표준출력과 표준오유를 위해 열려 진다. 자식쉘이 시작될 때 그것은 그의 어미로부터 표준입력, 표준출력, 표준오유를 계승한다. 그다음에 자식쉘은 표준입력이 건반에 들어 오게 하고 표준출력과 표준오유는 둘다 badstuff 파일에 나가게 한다. 이때 자식쉘은 >연산자를 처리한다. stdout를 파일 goodstuff에 대입한다. 출력은 goodstuff로 나가고 오유는 badstuff로 나간다. 그림 13-4에서 보여 준다.

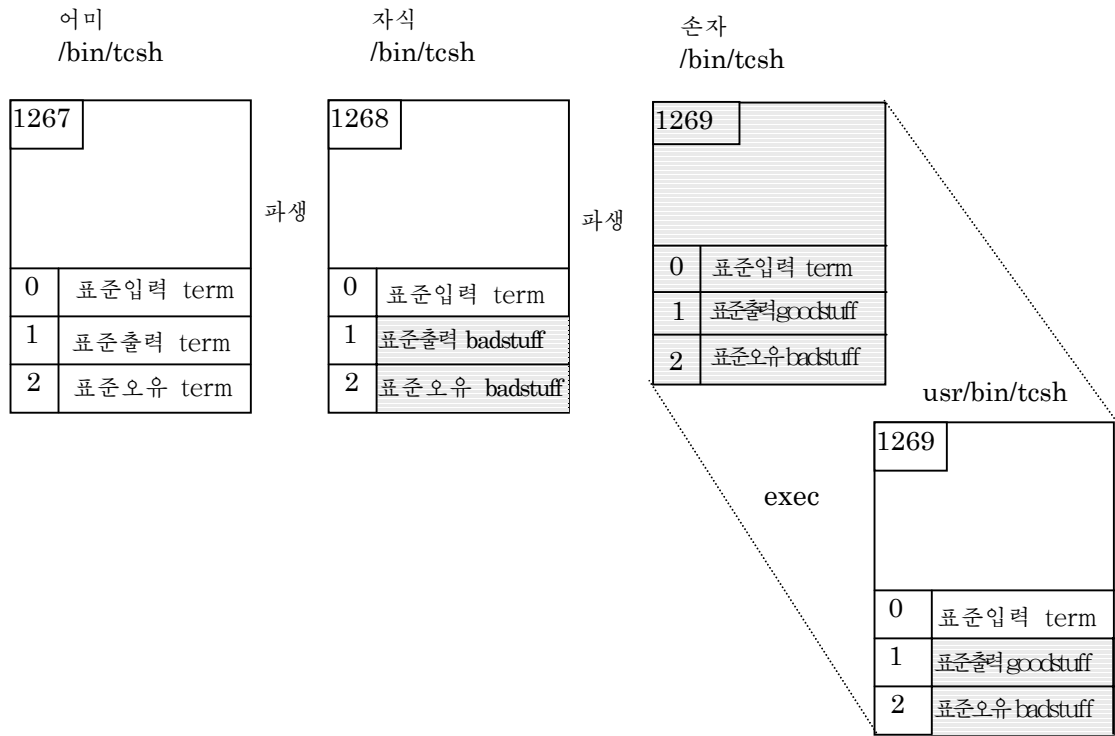



그림 13-4. 표준출력과 표준오유의 분리

noclobber변수 특수한 TC쉘내부변수 noclobber는 설정될 때 방향바꾸기에 의해 파일이 지워 지는것을 막는다. 표 13-16에 Noclobber변수를 주었다.

표 13-16. noclobber변수

noclobber가 설정되지 않았다.	파일이 존재한다.	파일이 존재하지 않는다.
command>file	파일이 덧 씌여 진다.	파일이 작성된다.
command>>file	파일이 추가된다.	파일이 작성된다.
noclobber가 설정된다.		
command>file	오유통보	파일이 작성된다.
command>>file	파일이 추가된다.	오유통보
noclobber무시		
command>!file	noclobber가 설정되면 이 지령에 대한 그의 영향을 무시하고 파일을 열거나 자르고 지령의 출력을 파일로 방향바꾸기한다.	
command>>!file	noclobber변수를 무시한다. 파일이 존재하지 않으면 파일을 작성하고 지령으로부터 출력을 파일에 추가한다.	

실례 13-60

```
1.  > cat filex
     abc
     123

2.  > date >filex
3.  > cat filex
     Tue Mar 18 11:51:04 pst 2001

4.  > set noclobber
5.  > date >filex
     filex: File exits.
6.  > ls >! filex       # Override noclobber   for this command only
     > cat filex
     abc
     ab1
     dir
     filex
     plan.c

7.  > ls > filex
     filex:   File   exist.

8.  > date >> xxx
     xxx: No such file or directory.

9.  > date >> ! xxx         # Override noclobber   for this command        only
10.   > unset noclobber # Turn off noclobber    permanently
```

설명

1. filex의 내용을 현시장치에 현시한다.
2. date지령의 출력을 filex로 방향바꾸기한다. 파일을 자르고 그의 초기내용들우에 겹쳐쓰기한다.
3. filex의 내용을 현시한다.
4. noclobber변수를 설정한다.
5. filex는 이미 존재하고 noclobber가 설정하였기때문에 쉘은 파일이 존재하며 그우에 겹쳐쓰기할수 없다고 통보한다.
6. ls의 출력은 >!연산자가 noclobber의 영향을 무시하기때문에 filex에로 방향바꾸기된다.
7. >!기호의 영향은 림시적이다. 그것은 noclobber를 해제하지 못하며 다만 리용된 지령을 위하여 noclobber를 무시한다.
8. noclobber가 설정되면 존재하지 않는 파일에로 date지령의 출력을 방향바꾸기하며 추가하려고 할 때에는 오유통보문을 현시한다.
9. noclobber변수를 >>방향바꾸기기호에 붙은 감탄부호로 무시한다.
10. noclobber변수의 설정을 해제한다.

제 7 절. 변수

tcsh변수는 오직 문자렬이나 문자렬모임만을 가진다. 일부 변수들은 쉘에 내장되여 있고 noclobber나 filec변수와 같이 그것들을 절환하여 설정할수 있으며 path변수와 같은 다른 변수들에는 문자렬값으로 대입한다. 또한 자체의 변수들을 만들수 있고 그것들을 문자렬이나 지령들의 출력으로 대입할수 있다. 변수이름들은 임의로 지을수 있는데 수자, 문자 그리고 밑줄로 된 20개의 문자까지 포함할수 있다.

두가지 형태의 변수 즉 국부변수와 환경변수가 있다. 변수의 유효범위는 그의 가시성을 가지고 있다. 국부변수는 그것을 정의한 쉘에서만 유효하다. 환경변수의 범위는 흔히 전역적이라고 한다. 그것들의 범위는 쉘과 이 쉘로부터 파생된 모든 프로쎄스들이다. 만일 국부변수를 set－r로 만들면 그것은 읽기전용으로 되여 변화시키거나 해제할수 없다.

화페기호（$）는 변수이름 앞에 있을 때 쉘이 그 변수의 값을 추출하라고 알려 주는 특수메타문자이다. echo지령은 변수가 인수로 주어질 때 쉘이 지령행을 처리하고 변수바꿔넣기를 진행한 다음에 그 변수의 값을 표시한다.

특수표기 $?는 변수이름앞에 붙을 때 변수가 설정되였는지 알수 있게 한다. 하나가 귀환되면 그것은 참 즉 변수가 설정되였다는것을 의미한다. 령이 귀환되면 그것은 거짓 즉 변수가 설정되지 않았다는것을 의미한다.

실례 13-61

```
1.  > set autologout
2.  > set history s 50
3.  > set name = George     '
4.  > set machine = 'uname -n'
5.  > echo $?machine
     1
6.  echo   $?blah
    0
```

설명

1. tcsh내부변수는 autolog out변수를 설정하여 비능동상태로 지적된 시간이 지난 다음에 자동적으로 등록탈퇴하게 할수 있다.
2. 현시되는 사건들의 수를 조종하기 위하여 내부변수 history를 50으로 설정한다.
3. 사용자정의변수 name을 George로 설정한다.
4. 사용자정의된 변수 machine을 UNIX지령의 출력으로 설정힌다. 지령은 거꿀인용부호안에 놓여 쉘이 지령바꿔넣기를 하라고 알려 준다.
5. $?는 변수이름앞에 붙어 변수가 설정되였는가를 검사한다. 검사결과가 1(참)이므로 변수는 설정되였다.
6. $?는 0(거짓)을 만든다. 변수는 설정되지 않았다.

1. 변수의 값인쇄

echo지령 echo내부지령은 그의 인수를 표준출력에로 인쇄한다. echo는 많은 탈출

문자들을 리용할수 있게 한다. 탈출문자들은 타브, 행바꾸기문자, 페지바꾸기 등으로 해석되고 현시된다. 표 13-17에서 echo추가선택들과 탈출문자들을 보여 준다.

T C쉘은 BSD와 SVR4의 형식을 리용하지만 내부변수 echo-style를 리용하여 echo지령의 방식을 수정할수 있게 한다. 실례로 echo-style=bsd로 설정한다. 표 13-18에서 echo를 위한 사용지도서페지를 보여 준다.

표 13-17. echo추가선택과 탈출문자

추가선택	의 미
-n	출력행의 끝에서 행바꾸기문자를 억제한다.
탈출문자	
\ a	종울리기
\ b	뒤방향지우기
\ c	행바꾸기문자가 없이 행을 인쇄한다.
\ f	페지바꾸기
\ n	행바꾸기문자
\ r	복귀
\ t	타브
\ v	수직타브
\ \	거꿀빗선
\ nnn	ASCII코드가 nnn(8진수)인 문자

표 13-18. echo_style변수

bsd	첫번째 인수가 –n이면 행바꾸기문자가 억제된다.
sysv	출력문자렬에서 탈출문자들을 확장한다.
both	-n과 탈출문자들이 둘다 리용될수 있다(기정으로).
nonc	sysv나 bsd를 인식하지 않는다.

실례 13-62

```
1. > echo The username is $LONGNAME.
   The username is ellie.
2. > echo "\ t\ tHello there\ c"
   Hello there>
3. >echo –n   "Hellow there"
   Hello   there$
4. > set echo_style=none
5. > echo   "\ t\ tHello there\ c"
   -n \ t\ tHello there\ c
```

설명

1. echo지령은 지령행에 있는 인수들을 현시장치에 인쇄한다. 변수바꿔넣기는 echo지령이 실행되기전에 쉘에 의해 수행된다.
2. echo지령은 기정으로 C프로그람작성언어의것들과 류사한 탈출문자들을 가지고 있으며 SVR4판본의 echo에서 사용된다. >는 쉘재촉문이다.
3. −n추가선택을 가진 echo는 행바꾸기를 하지 않고 문자렬을 표시한다.
4. echo_style변수에 값 none을 대입한다. BSD-n추가선택이나 SVR4 탈출문자들을 리용할수 없다.
5. 새로운 echo형식으로 문자렬들이 표시된다.

printf지령 printf의 GNU판본은 인쇄된 출력을 형식화하는데 리용될수 있다. 그것은 C의 printf함수와 같은 방법으로 형식화된 문자렬을 인쇄한다. 형식은 인쇄될 출력을 어떻게 보겠는가를 서술하는 형식지령을 포함하는 문자렬로 이루어 진다. 형식지령은 지적자(diouXfeEgGcs)가 뒤따르는 %로 지정되는데 여기서 %f는 류동소수점수를 표시하고 %d는 전체(소수) 수를 표현한다. printf지적자의 완전목록과 그것들의 리용법을 보기 위하여 지령행재촉문에서 printf--help라고 입력한다. 어떤 판본의 printf를 리용하는가를 보기 위하여 printf—version을 입력한다. 만일 bash 2.X를 리용하면 내부 printf지령은 /usr/bin에서 실행가능한 판본과 같은 형식을 리용한다.

형식

```
printf format [argument...]
```

실례 13-63

```
printf "%10.2f%5d\ n" 10.5 25
```

표 13-19. printf지령에 대한 형식지적자

형식지적자	값
\ "	2중인용부호
\ 0NNN	NNN이 0부터 3사이의 수자들을 나타내는 8진수문자
\ \	거꿀빗선
\ a	경고음내기
\ b	뒤방향지우기
\ c	출력을 더이상 내보내지 않는다.
\ f	페지바꾸기
\ n	행바꾸기문자
\ r	행복귀
\ t	수평타브

(표계속)

형식지적자	값
\ v	수직타브
\ xNNN	NNN이 1부터 3사이의 수자들을 나타내는 16진수문자
%%	단일%
%b	\ 탈출문자를 가진 문자렬로서 ARGMENT를 해석한다.

실례 13-64

1. > **printf –version**
   printf (GNU sh-utils) 1.16
2. > **printf "The number is So.2f\ " 100**
   The number is 100.00
3. > **printf "Ss-20s%-15s%10.2f\ n" "Jody" "Savage" 28**
   Jody   Savage   28.00
4. > **printf " |°S-20s |%-15s |Ssl0.2t(\ n" "Jody" "Savage" 28**
   |Jody |Savage | 28.00|
5. > **printf "%s's average was %.1f%%.\ n" "Jody"**
   $(((80+70+90)/3))
   Jody's average was 80.0%.

설명

1. printf지령은 GNU판본을 인쇄한다. 그것은 /usr/bin에 있다.
2. 인수 100이 소수점의 오른쪽 두자리만 류동소수점으로 인쇄한다. 이것은 형식문자렬에서 형식지정 %.2f로 지정한다. C에서와는 달리 인수들을 분리하는 반점이 없다.

3,4. 형식문자렬은 3개의 변환이 진행되도록 지정한다. 첫번째는 %- 20s(왼쪽끝맞추기한 20개 문자로 된 문자렬)이고 다음은 %-15s(왼쪽끝맞추기한 15개 문자로 된 문자렬)이며 마지막은 %10.2f(오른쪽끝맞추기한 10개문자로 된 류동소수점수인데 그중에서 한개 문자는 점이고 나머지 2개 문자는 소수점 오른쪽에 있는 2개 수자이다.)이다. 매 인수는 대조하는 %기호의 순서로 형식화되여 문자렬 Jody는 첫번째 %에 대조되고 문자렬 Savage는 두번째 %에 대조되고 수 28은 마지막 %기호에 대조된다. 수직선이 리용되여 마당의 너비를 표시한다.

5. printf지령은 문자렬 Jody와 산수연산확장의 결과를 형식화한다(632페지의 《산수연산확장》을 참고). 2개의 퍼센트 (%%) 기호는 하나의 퍼센트 기호 (%) 를 인쇄하기 위해 필요하다.

대괄호와 변수 대괄호는 변수를 그뒤에 놓이는 임의의 물음표들과 분리시킨다. 이것들을 리용하여 문자렬을 변수의 끝에 련결할수 있다.

실례 13-65

1. > set var = net
   > echo $var
   net

2. > **echo $varwork**
   varwork: Undefined variable.

3. > **echo ${var}work**
   network

설명

1. 변수이름주위에 있는 대괄호는 변수를 그뒤에 있는 문자와 분리시킨다.
2. varwork변수는 정의되지 않았다. 쉘은 오유통보문을 인쇄한다.
3. 대괄호는 변수를 그에 붙은 문자들로부터 보호한다. $var는 확장되며 문자렬 work를 추가한다.

2. 국부변수 (가시성과 이름짓기)

국부변수들은 그것들이 작성된 쉘에만 알려 진다. 국부변수가 .tcshrc파일에서 설정되면 변수는 새로운 TC쉘이 시동될 때마다 재설정된다. 습관상 국부변수는 소문자로 이름 짓는다.

국부변수설정 만일 대입되는 문자렬이 한개이상의 단어로 되여 있으면 그것을 인용부호안에 넣어야 한다. 그렇지 않으면 변수에 첫번째 단어만이 대입된다. 갈기부호주위에 공백이 있을수 있는데 한쪽에 공백이 있으면 다른 쪽에도 있어야 한다.

실례 13-66

1. > **set round = world**
2. > **set name = "Santa Claus"**
3. > **echo $round**
   world
4. > **echo $name**
   Santa Claus
5. > **tcsh** *# Start a subshell*
6. > **echo $name**
   name: Undefined variable.

설명

1. 국부변수 round에 값 world를 대입한다.

2. 국부변수 name에 값 Santa Claus가 대입한다. 2중인용부호는 쉘이 Santa 와 Claus사이에 공백을 해석하지 않도록 한다.
3. 변수앞에 놓여 있는 화페기호는 쉘이 변수바꿔넣기를 진행하게 하는데 즉 변수에 기억된 값을 뽑아 내게 한다.
4. 변수바꿔넣기가 수행된다.
5. 새로운 TC쉘(자식쉘이라고 한다.)프로쎄스가 시작된다.
6. 자식쉘에서 변수 name이 정의되지 않았다. 그것은 국부변수로서 어미쉘에서 정의되였다.

읽기전용변수 읽기전용변수는 일단 설정되면 변경될수도 해제할수도 없는 국부변수이다. 만일 변경시키거나 해제하려고 하면 오유통보문이 나온다. 환경변수는 읽기전용으로 만들수 없다.

실례 13-67

1. > **set -r name = Tommy**
2. > **unset name**
   unset: $name is read-only.
3. > **set name = Danny**
   set: $name is read-only

set지령 set지령은 이 쉘에서 설정된 모든 국부변수들을 인쇄한다.

실례 13-68

```
(The  Command Line)
     > set
addsuffix
argv          ()
cwd           /home/jody/meta
dirstack      /home/ellie/meta
echo_style both
edit
gid        501
grupe      ellie
history    500
home       /home/ellie
I          /etc/profile.d/mc.csh
Owd        /home/ellie
Noclobber
Path       (/usr/sbin/sbin/usr/local/bin/bin /usr/bin/usr/X11R6/bin)
Prompt     [%n@%m     %c]   #
Prompt2    %R
Prompt3    CORRECT>%R    (y/n/e/a)?
Savedirs
```

```
Shell        /bin/tcsh
Shlvl        2
Status       0
Tcsh         6.07.09
Term         xterm
User         ellie
Version   tcsh   6.09.09 (Astron)   1998-09-16   (sparc-sun-solaris)
Option   8b,nls,dl,al,rh,color
```

설명

이 쉘을 위해 설정된 모든 국부변수들이 인쇄된다. history, dirstack, noclobber 변수들가운데서 많은것이 tcshrc파일에 설정된다. argr, cwd, shell, term, usr, version, status와 같은 변수들은 이미 설정된 내부변수들이다.

내부국부변수 쉘은 변수들의 정의와 함께 미리 정의된 변수들을 많이 가지고 있다. 일부 변수들은 절환된다. 실례로 noclobber를 설정하면 변수는 설정되고 리용할 수 있다. noclobber를 해제하면 그것을 리용하지 못한다. 일부 변수들은 설정될 때 정의를 요구한다. 내부변수들은 대개 그것들을 대화형 T C쉘과 tcsh스크립트들을 위하여 리용할수 있으면 .tcshrc파일에서 설정한다. 이미 론의된 내부변수들에는 noclobber, cdpath, history, rmstar, noglob 등이 있다. 표 13-24에서 완성된 목록을 주었다.

3. 환경변수

환경변수들은 흔히 전역변수라고 부른다. 이 변수들은 작성한 쉘에서 정의되고 그 쉘로부터 파생된 모든 쉘들에 의해 계승된다. 비록 환경변수들이 자식쉘에 의해 계승되지만 자식쉘에서 정의된 변수들은 어미쉘로 넘겨 지지 않는다. 계승은 어미로부터 자식에로 진행되고 반대로는 되지 않는다. 습관상 환경변수들은 대문자로 이름 짓는다.

실례 13-69

```
(The Command Line)
1.  > setenv TERM wyse
2.  > setenv PERSON "Joe Jr."
3.  > echo $TERM
     wyse
4.  > echo $PERSON
     Joe Jr.
5.  > echo $$        # $$ evaluates to the PID of the current shell 206

6.  > tcsh                   # Stark a subshell
7.  > echo $$
     211
8.  > echo $PERSON
     Joe Jr.
```

```
9.  > setenv PERSON "Melly Nerd"
10.  > echo $PERSON
    Nelly Nerd.
11.  > exit                  # Exit the subshell

12.  > echo $$
    206
13.  > echo $PERSON          # Back in parent shell
    Joe Jr
```

설명

1. 환경변수 TERM을 wyse말단에로 설정한다.
2. 사용자정의된 변수 PERSON을 Joe Jr로 설정한다. 인용부호는 공백을 보호하는데 리용된다.
3. 변수이름앞에 붙은 화페기호($)는 쉘이 변수의 내용들을 해석하게 한다. 이것을 변수바꿔넣기라고 한다.
4. 환경변수 PERSON의 값을 인쇄한다.
5. $$변수에는 현재쉘의 PID가 들어 있다. PID는 206이다.
6. tcsh지령은 자식쉘이라는 새로운 T C쉘을 기동한다.
7. 현재쉘의 PID가 인쇄된다. 이 쉘은 새로운 T C쉘이기때문에 그것은 다른 PID번호를 가지고 있다. PID는 211이다.
8. 환경변수 PERSON은 새로운 쉘에 의해 계승되였다.
9. PERSON변수는 Nelly Nerd로 재설정된다. 이 변수는 임의의 쉘들에 의해 계승된다.
10. PERSON변수의 새로운 값이 인쇄된다.
11. 이 쉘은 탈퇴한다.
12. 초기쉘이 실행된다. 그것을 증명하기 위해 PID 206이 인쇄된다. 그것은 자식쉘이 시작되기전의 값과 같다.
13. PERSON변수는 그의 초기값을 가지고 있다.

환경변수현시 printenv(BSD)와 env(SVR4)지령은 이 쉘과 그의 자식쉘을 위해 설정된 모든 환경변수들을 인쇄한다. setenv지령은 T C쉘의 UCB와 SVR4판본에서 변수들과 그들의 값을 인쇄한다.

실례 13-70

```
> env or printenv or setenv
USERNAME=root
COLORTERM=rxvt-xpm
HISTSIZE=1000
HOSTNAME=homebound
LOGENAME=ellie
HiISTFILESIZE=1000
MALL=/var/spool/mail/ellie
```

```
MACHTYPE=i386
COLORFGBG=0;default;15
TERM=xterm
HOSTTYPE=i386-unix
PATH=/usr/sbin:/sbin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin;/home/ellie/bin/
;root/bash-2.03/:/usr/X11R6/bin/:/home/ellie/bin;/root/bash-2.03/:/usr/X11R6/bin
HOME=/root
SHELL=/bin/bash
PS1=[\ u@\ h\ W]\ $
USER=ellie
VENDOR=intel
GROUP=ellie
HOSTDISPLAY=homebound:0.0
DISPLAY=:0.0
HOST=homebound
OSTYPE=unix
WINDOWID=37748738
PWD=/home/ellie
SHLVL=6
_=/usr/bin/env
```

설명

환경변수들은 이 대화조종을 위해 설정하며 이 쉘에서 시작된 모든 프로쎄스들은 내부지령들중 어느 하나 즉 env나 printenv를 리용하여 현시한다. 많은 응용프로그람들은 환경변수들의 설정을 요구한다. 실례로 mail지령은 사용자의 우편스풀러의 위치로 설정된 MAIL변수를 가지며 xterm프로그람은 어느 비트매프현시말단을 리용하겠는가를 결정하는 DISPLAY변수를 가지고 있다. 이 프로그람들중 어느것을 실행할 때 그 개별적인 변수에 있는 값들을 그 프로그람들에로 넘긴다.

제 8 절. 배렬

1. 배렬이란 무엇인가

T C쉘에서 배렬은 단순히 공백이나 타브건에 의해 분리되는 단어들의 목록이므로 괄호안에 있다. 배렬의 요소들은 1에서 시작하는 보조스크립트들에 의해 번호가 붙는다. 보조스크립트를 위한 배렬요소가 없다면 통보문 Subscript out of range를 현시한다. 지령바꿔넣기도 배렬을 작성한다. $#표기가 배렬이름앞에 있으면 배렬에 있는 요소들의 번호가 표시된다.

실례 13-71

```
1.  > set fruit = ( apples pears poaches plums )
2.  > echo $fruit
```

apples pears peaches plums

3. **> echo $fruit[l]** *# Subscripts start at 1 appies*
4. **> echo $fruit[2-4]** *# Prints- the 2nd,3rd, and 4th elements pears peaches plums*
5. **> echo $fruit[6]**
   Subscript out of range
6. **> echo $fruit[*]** *# Prints all elements of the array apples pears peaches plums*
7. **> echo $#fruit** *# Prints the number of elements 4*
8. **> echo $%£ruit** *# Prints the number of characters in the* list *23*
9. **> echo $fruit[$#fruit]** *# Prints the last element*
   plums
10. **> set fruit[2] = bananas** *# Reassigns the second element*
    > echo $fruit
    apples bananas peaches plums
11. **> set path = (~ /usr/bin/usr/usr/local/bin .)**
    > echo $path
    /home/jody/ellie/usr/bin/usr/usr/local/bin .
12. **> echo $path[1]**
    /home/jody/ellie

설명

1. 단어목록은 괄호안에 들어 있으l며 매 단어는 공백으로 분리된다. 배렬은 fruit라고 부른다.
2. fruit배렬에서 단어들이 인쇄된다.
3. fruit배렬의 첫 요소가 인쇄된다. 보조스크립트들은 1부터 시작한다.
4. 단어목록의 둘째, 셋째, 넷째 요소들이 인쇄된다. 횡선은 범위를 지정하게 한다.
5. 배렬은 6개 요소를 가지지 않는다. 보조스크립트는 범위에서 벗어 난다.
6. fruit배렬의 모든 요소들이 인쇄된다.
7. 배렬앞에 $#기호를 리용하여 배렬에 있는 요소들의 번호를 얻는다. fruit배렬에는 4개 요소들이 있다.
8. 변수나 배렬앞에 있는 $%기호는 단어목록에 있는 문자들의 개수를 인쇄한다.
9. 보조스크립트 $#fruit가 배렬에 있는 요소들의 전체 개수이므로 그 값을 배렬의 첨수값 즉 [$#fruit]로 리용하면 fruit배렬의 마지막요소를 인쇄한다.
10. 배렬의 두번째 요소에 새로운 값을 대입한다. 배렬은 그의 교체된 값 bananas로 인쇄된다.

11. path변수는 지령을 찾는데 리용되는 등록부들의 특수한 T C쉘배렬이다. 배렬을 만들 때 경로의 개별적요소들은 호출되거나 변경될수 있다.
12. path의 첫 요소를 인쇄한다.

shift지령과 배렬 만일 내부 shift지령이 인수로서 배렬이름을 가진다면 그것은 배렬의 첫 요소를 왼쪽으로 자리밀기하고 배렬의 길이가 하나 감소된다. shift지령은 인수가 없이 내부 argv배렬의 첫 요소를 자리밀기한다.

표 13 -20. 변수변경자

특수변경자	실 례	의 미
$?	$?var	Var를 설정하면 1을 복귀한다. 아니면 0을 복귀한다.
$#	$#var	var에 있는 단어들의 개수를 복귀한다.
$%	$%var	var에 있는 문자들의 개수를 복귀한다.

실례 13-72

1. > **set names = (Mark Tom Liz Dan Jody)**
2. > **echo $names**
   Mark Tom Liz Dan Jody
3. > **echo $names[l]**
   Mark
4. > **shift   names**
5. > **echo $names**
   Tom Liz Dan Jody
6. > **echo $names[l]**
   Tom
7. > **set days = (Monday Tuesday)**
8. > **shift days**
9. > **echo $days**
   Tuesday
10. > **shift days**
11. > **echo $days**
12. > **shift days**
    shift: no more words.

설명

1. 배렬은 names라고 부른다. 괄호안에 있는 단어들의 목록을 대입한다. 매 단어는 공백으로 분리된다.
2. 배렬이 인쇄된다.

3. 배렬의 첫 요소가 인쇄된다.
4. 배렬은 한개의 요소에 의해 왼쪽으로 밀리운다. 단어 Mark는 밀리워 없어 진다.
5. 배렬은 밀기한후에 한개의 요소가 감소된다.
6. 밀기한후 배렬의 첫 요소는 Tom이다.
7. days라고 부르는 배렬을 만든다. 그것은 2개의 요소 Monday와 Tuesday를 가지고 있다.
8. days배렬을 하나 왼쪽으로 밀기한다.
9. 배렬이 인쇄된다. Tuesday만이 남는다.
10. days배렬은 다시 자리밀기된다. 배렬이 빈다.
11. days배렬이 빈다.
12. 이때 자리밀기하면 쉘이 빈 배렬로부터 요소를 밀수 없다고 표시하는 오유통보를 보내도록 한다.

문자렬로부터 배렬만들기 인용부호안에 있는 문자렬로 단어목록을 작성하려고 할수 있다. 이것은 괄호안에 문자렬변수를 넣어 실현한다.

실례 13-73

1. **> set name = "Thomas Ben Savage"**
   > echo $name[l]
   Thomas Ben Savage
2. **> echo $naine[2]**
   Subscript out of range
3. **> set name a ($name)**
4. **> echo $name[l] $name[2] $name[3]**
   Thomas Ben Savage

설명

1. 변수 name에 문자렬 Thomas Ben Savage를 대입한다.
2. 배렬로 취급할 때 오직 한개 요소 즉 전체 문자렬만이 있다.
3. 변수는 괄호안에 넣어 져 name이라고 하는 단어들의 배렬을 만든다.
4. 새로운 배렬의 3개 요소들을 현시한다.

제 9 절. 특수변수와 변경자

한개의 문자로 이루어 진 여러개의 변수들이 T C쉘에 내장되여 있다. 문자앞에 붙어 있는 $는 변수를 해석하게 한다(표 13-21).

표 13-21. 변수와 그것들의 의미

변 수	실 례	의 미
$?var	echo $?name	변수가 설정되였으면 1을, 아니면 0을 복귀한다.
$#var	echo $#fruit	배렬에 있는 요소들의 수를 인쇄한다.
$%var	echo $%name	변수나 배렬에 있는 문자들의 수를 인쇄한다.
$$	echo $$	현재쉘의 PID를 인쇄한다.
$<	set name = $<	사용자로부터 행바꾸기문자까지 입력행을 받아 들인다.
$?	echo $?	$status와 같다. 마지막지령의 탈퇴상태를 포함한다.
$!	kill $!	배경에 넣어 진 마지막과제의 프로세스 ID번호를 포함한다.

실례 13-74

```
1.  > set   num
    > echo   $?num
    1

2.  > echo $path
    /home/jody/ellie /usr /bin /usr/local/bin
    >   echo   $#path
    3
3.  > echo   $$
    245

    > tcsh                    # Start a subshell
    >echo   $$
    248

4.  > set name = $<
    Chistry Campbell

    > echo   $name
    Christy   Campbell

5.  > set name =”$<”
    Chistry Campbell
    > echo $name
    Chistry Campbell
```

설명

1. 변수 num을 빈값으로 설정한다. 변수앞에 붙어 있는 $?는 변수가 설정(빈값이나 어떤 값으로)되였다면 1과 같고 변수가 설정되지 않았다면 령과 같다.
2. path변수를 인쇄한다. 그것은 3개의 요소를 가진 배렬이다. 변수앞에 붙어 있는 $#는 배렬에 있는 요소들의 수를 추출하고 인쇄한다.

3. $$는 현재프로세스 즉 이 경우에는 C쉘의 PID이다.
4. $<변수는 사용자로부터 처음에 오는 첫번째 공백이나 행바꾸기문자까지 입력단어를 받아서 그 단어를 name변수에 기억시킨다. name변수의 값을 현시한다.
5. $<변수는 인용부호(2중인용부호)안에 있을 때 사용자로부터 행바꾸기문자까지 입력행을 받는데 이때 행바꾸기문자는 포함하지 않는다. 그리고 그 행을 name변수에 기억시킨다. name변수의 값을 현시한다.

1. 경로이름변수변경자

경로이름은 변수에 대입되면 특수T C쉘확장자를 거기에 추가하여 경로이름변수를 조작할수 있다. 경로이름을 4개의 부분 즉 head, tail, root, extension으로 나눈다. 경로이름변경자들과 그것들의 기능을 표 13-22에 주었다.

표 13-22. 경로이름변경자 (set pn = /home/ellie/prog/check.c)

변경자	의 미	실 례	결 과
:r	뿌리	echo $pn:r	/home/ellie/prog/check
:h	머리부	echo $pn:h	/home/ellie/prog
:t	꼬리	echo $pn:t	check.c
:e	확장	echo $pn:e	c
:g	대역	echo $p:gt	(실례 13-75를 참고)

실례 13-75

1. > **set pathvar = /home/danny /program.c**
2. > **echo $pathvarsr**
   /home/danny/program
3. > **echo $pathvar:h**
   /home/danny
4. > **echo $pathvar:t**
   program.c
5. > **echo $pathvar!e**
   c
6. > **set pathvar = (/home/*)**
   echo $pathvar
   */home/Jody /home/local /home/lost*round /home/peri /hoine/tmp*
7. > **echo $pathvar:gt**
   jody local lost+found peri tmp

설명

1. 변수 pathvar를 /home/danny/program.c로 설정한다.
2. ：r를 변수에 추가하면 확장자를 현시할 때 없어 진다.
3. ：h를 변수에 추가하면 경로의 머리부가 현시된다. 즉 경로의 마지막요소가 지우기된다.
4. ：t를 변수에 추가하면 경로의 꼬리끝(마지막요소)이 현시된다.
5. ：e를 변수에 추가하면 확장자가 현시된다.
6. 변수를 /home/*에로 설정한다. 별표는 현재등록부에 있는 /home/으로 시작하는 모든 경로이름들로 확장된다.
7. ：gt를 변수에 추가하면 매 경로요소들의 꼬리끝이 현시된다.

2. 대소문자변경자

특수리력변경자를 리용하여 변수에 있는 문자들의 선택을 변경시킨다.

표 13-23. 선택변경자

변경자	기 능
:u	단어에 있는 첫번째 소문자를 대문자로 한다.
:l	단어에 있는 첫번째 대문자를 소문자로 한다.
:g	매 단어에 변경자를 한번 적용한다.
:a	한개 단어에 대해 가능한 변경자를 적용한다.

실례 13-76

1. **> set name = nicky**
   > echo $name:u
   Nicky
2. **> set name = (nicky jake)**
   > echo $name:gu
   Nicky Jake
3. **> echo $name:agu**
   NICKY JAKE
4. **> set name = (TOMMY DANNY)**
   > echo $name:agl
   tommy danny
5. **> set name = "$name:agu"**
   >echo $name
   TOMMY DANNY

설명

1. ：u를 변수에 추가하면 그 값에 있는 첫 문자가 대문자로 된다.
2. :gu를 변수에 추가하면 값목록에서 매 단어의 첫 문자가 대문자로 된다.
3. :agu를 변수에 추가하면 그 값에서 모든 문자가 대문자로 된다.
4. :agl을 변수에 추가하면 그 값에서 모든 문자가 소문자로 된다.
5. 변수는 그 목록에서 대문자로 표시된 목록에 있는 모든 문자로 재설정된다.

제 10 절. 지령바꿔넣기

1. 거꿀인용부호

문자렬이나 변수는 거꿀인용부호안에 지령을 넣어 UNIX지령의 출력으로 대입될수 있다. 이것을 지령바꿔넣기라고 한다(건반에서 거꿀인용부호는 보통 ~문자아래에 있다.). 만일 지령의 출력을 변수에 대입하면 그것은 단어목록이나 배렬로서 기억된다(아래의《단어목록과 지령바꿔넣기》에서 참고).

실례 13-77

1. **> echo The name of my machine is 'unams -n*.**
   The name of my machine is stardust.

2. **> echo *The present working directory is 'pwd'.***
   The present working directory is /home /sta.rdu.st /john.

3. **> set d = 'date'**
   > echo $d
   Tue Mar 28 14:24:21 PDT 2001

설명

1. UNIX지령 uname-n은 거꿀인용부호안에 있다. 쉘이 거꿀인용부호를 발견할 때 그 안에 있는 지령 uname-n을 실행하고 지령 stardust의 출력을 문자렬에 치환한다. echo지령이 그의 인수를 표준출력으로 인쇄할 때 콤퓨터의 이름은 그 인수중 하나로 된다.
2. UNIX지령 pwd는 쉘에 의해 실행되고 출력은 문자렬안의 그 자리에 교체된다.
3. 국부변수 d에 date지령의 출력을 대입한다. 출력은 단어들의 목록으로 기억된다.

단어목록과 지령바꿔넣기 지령이 거꿀인용부호안에 있고 변수에 대입되면 결과값은 배렬（단어목록）이다. 배렬의 매 요소는 보조스크립트를 배렬이름에 추가하여 호출할수 있다. 보조스크립트는 1에서 시작한다. 배렬에서 단어들의 개수보다 더 큰 보조스크립트를 리용하면 TC쉘은 subscript out of range를 인쇄한다. 지령의 출력이 한개이상의 행으로 이루어 지면 행바꾸기문자들은 매 행에서 빠져 한개의 공백으로 교체된다.

배렬이 만들어 진 다음 내부 shift지령을 리용하여 단어번호 1에서 시작하는 단어들을 지우기할수 있다. 일단 단어가 밀리우면 그것은 회복될수 없다.

실례 13-78

1. > **set d = 'date'**
   > echo $d
   Tue Mar 28 14:04:49 *PS"P 2001*
2. > **echo $d[l-3]**
   Tue Mar 28
3. > **echo $d[6]**
   2001
4. > **echo $d[7]**
   Subscript out of range.
5. > **echo "The calendar for the month of March is -cal 3 2000'"**
   The calendar for month of March is March 3001 S M Tu W Th F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

설명

1. 변수 d에 UNIX date지령의 출력을 대입한다. 출력을 배렬로서 기억한다. 변수의 값을 현시한다.
2. 배렬의 첫 3개 요소들을 현시한다.
3. 배렬의 여섯번째 요소를 현시한다.
4. 배렬에는 7개의 요소가 없다. 쉘은 보조스크립트가 범위를 벗어 난다고 통보한다.
5. 출력은 한개이상의 행으로 된다. 매 행바꾸기문자는 공백으로 교체된다. 이것은 예견했던 출력이 아닐수 있다.

실례 13-79

1. > set machine = 'rusers | awk ‘/tom/{print $1}’ '
2. > **echo $roachine**
   dumbo baiabi dolphin
3. > echo $#machine
   3
4. > **echo $iBachine[$»Biaehine]**
   dolphin
5. > **echo $machine**
   dumbo bambi dolphin
6. > **shift $machine**
   > **echo $machine**
   bambi dolphin
7. > **echo $machine[l]**

bambi

8. > **echo** $#machine
2

설명

1. rusers지령의 출력은 awk로 파이프된다. 정규식 tom을 찾으면 awk는 첫 마당을 인쇄한다. 이 경우에 첫 마당은 사용자 tom이 등록가입한 콤퓨터의 이름이다.
2. 사용자 tom이 3개의 콤퓨터에 등록가입된다. 콤퓨터들의 이름을 현시한다.
3. 배렬에서 요소들의 개수를 배렬이름에 $#를 붙여 호출한다. 배렬에는 3개의 요소가 있다.
4. 배렬의 마지막요소를 현시한다. 배렬에 있는 요소들의 개수 ($#machine) 를 보조스크립트로 리용한다.
5. 배렬을 현시한다.
6. shift지령은 배렬을 왼쪽으로 자리밀기한다. 배렬의 첫 요소가 없어 지고 보조스크립트의 번호를 1에서 시작하여 다시 달아 준다.
7. 자리밀기한후 배렬의 첫 요소를 현시한다.
8. 자리밀기한 다음에 배렬의 길이는 하나 감소된다.

제 11 절. 인용부호찍기

T C쉘은 어떤 특수한 의미를 가지는 메타문자들의 전체 모임을 가지고 있다. 사실상 문자나 수자가 아닌 건반에 있는 거의 모든 문자는 쉘에 대해서 어떤 특수한 의미를 가진다. 여기에 다음과 같은 부분적인 목록을 보여 주었다.

* ? [ ] % ~ ! & { } ( ) > < | ; : %

거꿀빗선과 인용부호를 리용하여 쉘이 메타문자들을 해석하지 않게 할수 있다. 거꿀빗선을 리용하여 단일문자를 탈퇴시키며 인용부호를 리용하여 문자들의 렬을 보호할수 있다. 인용부호를 사용하는데서 지켜야 할 다음과 같은 몇가지 일반규칙이 있다.

1. 인용부호들은 쌍을 이루며 한행에서 대조되여야 한다. 거꿀빗선은 인용부호가 다음행에서 대조되도록 행바꾸기문자를 행바꾸기로 사용되게 한다.
2. 단일인용부호는 2중인용부호를 보호하고 2중인용부호는 단일인용부호를 보호한다.
3. 단일인용부호는 리력문자 (!) 를 제외하고 모든 메타문자들이 해석되지 않게 보호한다.
4. 2중인용부호는 리력문자 (!), 변수바꿔넣기문자 ($), 거꿀인용부호(지령바꿔넣기에 리용된다.)를 제외한 모든 메타문자들이 해석되지 않게 보호한다.

1. 거꿀빗선

거꿀빗선은 단일문자를 인용하는데 리용되며 리력문자렬(!!, !string, !5)을 탈퇴시키는데 리용할수 있는 유일한 문자이다. 흔히 거꿀빗선은 행바꾸기문자를 탈퇴시키는데 리용한다. 특수한 tcsh변수 backslash_quote는 인용부호나 거꿀빗선을 인용하는데 리용될수 있지만 cs스크립트에서는 기능을 수행하지 못한다(실례13-80).

실례 13-80

1. **> echo Who are you?**
   echo: No match.

2. **> echo Who are you\ ?**
   Who are you?

3. **>echo This is a very, very long line and this is where**
   ? I break the line.
   This is a very, very long line and this is where I break the line.

4. **> echo "\ abc"**
   \ abc
   > echo '\ abc'
   \ abc
   > echo \ \ abc
   \ abc

5. **> echo 'I can't help it!'**
   Unmatched
6. **> set backslash_quote**
   > echo 'I can't help it!'
   I can't help it!

설명

1. 물음표는 파일이름확장에 리용한다. 그것은 단일문자를 대신한다. 쉘은 단일문자가 뒤따르는 y-o-u라는 현재등록부에서 파일을 찾고 있다. 등록부에 그 파일이 없기때문에 쉘은 No match로 대조되는것을 찾을수 없다고 통보한다.
2. 쉘은 물음표가 거꿀빗선에 의해 탈퇴되기때문에 그것을 해석하지 않는다.
3. 문자렬은 거꿀빗선으로 행바꾸기문자를 탈퇴시켜 다음행에서 계속된다.
4. 만일 거꿀빗선이 단일 혹은 2중인용부호안에 있으면 인쇄하고 인용부호안에 있지 않으면 거꿀인용부호는 자기자체를 무시한다.
5. 단일인용부호안에 있으면 거꿀인용부호는 인용부호문자처럼 무시된다. 즉 그것은 can't에 있는 단일웃반점을 보호하지 못한다. 쉘은 3개의 단일인용부호를 보고 한개의 인용부호가 대조되지 않는다고 통보한다.
6. tcsh변수 backslash_quote가 설정되면 거꿀빗선은 항상 거꿀빗선, 단일인용부호 그리고 2중인용부호를 인용한다.

2. 단일인용부호

단일인용부호는 같은 행우에서 대조되여야 하며 리력문자 (!) 를 제외하고 모든 메타문자들을 탈퇴시킨다. 쉘이 인용부호를 해석하기전에(그렇지만 거꿀빗선을 해석하기전은 아니다.) 리력을 평가하기때문에 리력문자는 보호되지 않는다.

실례 13-81

1. > **echo "I need $5.00'**
   I need $5.00

2. > cp filel file2
   > **echo 'I need $500.00 now!!'**
   echo 'I need $500.00 nowcp filel file2'

3. > **echo 'I need $500.00 now\ !\ !"**
   I need *$500.00 now!!*

4. > **echo 'This is going to be a long line so**
   Unmatched '.

5. > echo 'This is going to be a long line so \
   ? I used the 거꿀빗선 to suppress the newline"
   This is going to *be a long- line so*
   I used. the *거꿀빗선 to suppress the newline*

설명

1. 문자렬은 단일인용부호안에 있다. 리력문자(!)를 제외한 모든 문자들을 쉘해석으로부터 보호한다.
2. cp지령을 리용한 다음 echo프로그람을 실행한다. !!는 거꿀빗선을 리용함으로써 쉘해석으로부터 보호되지 않았기때문에 리력목록의 마지막지령이 재실행되고 cp지령은 문자렬의 부분으로 된다.
3. 리력문자 (!!) 를 거꿀빗선으로 인용하면 그것들은 리력바꿔넣기로부터 보호된다.
4. 인용부호는 같은 행우에서 대조되여야 한다. 그렇지 않으면 쉘은 unmatched라고 통보한다.
5. 행이 계속되면 거꿀빗선을 리용하여 행바꾸기문자를 탈퇴시킨다. 인용부호는 다음행의 끝에서 대조된다. 쉘이 행바꾸기문자를 무시하여도 echo지령은 무시하지 않았다 (?는 2차재촉문 prompt2이다.) .

3. 2중인용부호

2중인용부호는 대조되여야 하고 변수와 지령바꿔넣기를 허락하며 리력문자(!)를 제외하고 모든것을 보호한다. 거꿀빗선은 2중인용부호안에 있을 때 화페기호를 탈퇴시키지 못한다.

실례 13-82

1. **> set name = Bob**
   > echo "Hi $name"
   Hi Bob
2. **> echo "I don't have time."**
   I don't have time.
3. **> echo "WOW!!"** # Watch the history metacharacter! *echo "Wowecho "Wow!""*
4. **> echo "WOW\ !\ ! "**
   Wow ! !
5. **> echo "I need \ $5.00"**
   I need \ .00

설명

1. 극부변수 name에 값 Bob를 대입한다. 2중인용부호는 화페기호가 변수바꿔넣기로서 리용되게 한다.
2. 단일인용부호는 2중인용부호안에서 보호된다.
3. 2중인용부호, 단일인용부호는 감탄표를 쉘해석으로부터 보호하지 못한다. 내부 history지령은 2중인용부호로 시작하는 마지막지령을 찾는데 그 사건은 찾지 못하였다.
4. 거꿀빗선은 감탄표를 보호하는데 리용된다.
5. 거꿀빗선은 2중인용부호안에서 리용될 때 화페기호를 탈퇴시키지 못한다.

4. 2중인용부호와 단일인용부호의 조합

인용부호짝기규칙을 지키면 2중인용부호와 단일인용부호를 단일지령에서 다양한 조합으로 리용할수 있다.

실례 13-83

1. **> set name = Tom**
2. **> echo "I can't give $name" ' $5.00\ !\ !'**
   I can't give Tom $5.00!!
3. **> echo She cried, \ "0h help Me\ !\ !"' ', $name.**
   She cried, "Oh help me ! ! ", Tom.

설명

1. 국부변수 name에 Tom을 대입한다.
2. 단어 can’t에 있는 단일인용부호는 2중인용부호안에 있을 때 보호된다. 쉘은 $5.00에서 화페기호가 2중인용부호안에 있었다면 변수바꿔넣기를 진행한다. 그러나 문자렬 $5.00이 단일인용부호안에 있으므로 화페기호는 문

자 그대로 된다. 2중인용부호나 단일인용부호가 쉘해석으로부터 거꿀빗선을 보호할수 없기때문에 거꿀빗선으로 감탄부호를 보호한다.
3. 첫 대화인용부호는 거꿀빗선에 의해 보호된다. 감탄부호도 거꿀빗선에 의해서 보호된다. 마지막인용부호는 한조의 단일인용부호안에 놓인다. 단일인용부호는 2중인용부호를 보호한다.

5. 인용부호를 옳게 리용하기 위한 단계

보다 복잡한 지령에서 다음과 같은 단계들을 지키지 않으면 인용부호를 정확히 대조시키기가 힘들다 (부록 3을 참고).

1. UNIX지령과 그 문법을 잘 알아야 한다. 변수바꿔넣기를 하기전에 지령행에 값들을 넣어 요구되는 결과를 얻을수 있는가를 본다.

```
> awk –F: ‘/^Zippy Pinhead/{print “Phone is  “ $2}’ datafile
Phone is 408-123-4563
```

2. UNIX지령이 정확히 진행되였으면 변수들을 넣는다. 여기서 임의의 인용부호들을 없애거나 변화시키지 말아야 한다. 다만 변수들을 그것들이 표시하는 단어들대신에 넣어야 한다. 이 실례에서는 Zippy Pinhead를 $name으로 교체한다.

```
> set name = “Zippy Pinhead”
> awk –F: ‘/^$name/{print “Phone is “$2}’ datafile
```

3. 인용부호찍기는 다음과 같이 한다. 왼쪽에서 첫번째 단일인용부호로 시작하여 $name에 있는 화페기호앞에 닫는 단일인용부호를 삽입한다. 이때 한쌍의 인용부호가 대조된다.

```
> awk –F: ‘/^$name’/{print “Phone is “ $2}’ datafile
```

그다음 $name에서 마지막문자 e의 오른쪽뒤에 또 다른 단일인용부호를 넣는다. 이 인용부호는 닫긴 대괄호뒤에 있는 인용부호와 대조된다.

```
> awk –F: ‘/^’$name’/{print “Phone is “$2}’ datafile
```

왼쪽에서 시작하여 단일인용부호의 개수를 계산한다. 그 수는 짝수인 4이다. 매개 쌍의 단일인용부호안에 있는 모든것이 쉘에 의해 무시된다. 인용부호들은 다음과 같이 대조된다.

```
> nawk –F: ‘$1 ~ /’$name’/{print $2}’ filename
```

4. 마지막단계 : 변수들을 2중인용부호안에 넣는다. 매 변수들을 한조의 2중인용부호안에 넣는다. 2중인용부호들은 확장된 변수에서 공백을 보호한다. 실례로 Zippy Pinhead에서 공백이 보호된다.

```
> nawk –F: ‘$1 ~ / ‘ “$name” ‘/{pirnt $2}’ filename
```

6. 변수를 인용부호안에 넣기

:x와 :q변경자는 변수를 인용부호안에 넣어야 할 때 리용된다.

:q변경자로 인용부호찍기 :q변경자는 2중인용부호를 교체시키는데 리용된다.

실례 13-84

1. **> set name = "Daniel Savage"**
2. **> grep $name:q database**
   same as
3. **> grep "$nanr" database**
4. **> set food = "apple pie"**
5. **> set dessert = ($£ood "ice csftoi")**
6. **> echo $ftdessert**
   3
7. **> echo $dessert[l]**
   apple
8. **> echo $dessert[2]**
   pie
9. **> echo $dessert[3]**
   ice cream
10. **> set dessert = ($food:q "ice cream")**
11. **> echo $#dessert**
   2
12. **> echo $dessTt[l]**
   apple pie
13. > echo $deeawt[2]
   ice cream

설명

1. 변수에 문자렬 Danial Savage를 대입한다.
2. :q가 변수에 추가되여 변수가 인용부호안에 놓인다. 이것은 2중인용부호안에 변수를 포함시키는것과 같다.
3. 변수 $name을 둘러 싸고 있는 2중인용부호는 변수바꿔넣기가 진행되게 하지만 임의의 공백문자를 보호한다. 2중인용부호가 없이 grep프로그람은 Savage라는 파일과 database라는 파일에서 Daniel을 탐색한다.
4. 변수 food에 문자렬 apple pie를 대입한다.
5. 변수 dessert에 apple pie와 ice cream으로 이루어 진 배렬 (단어목록) 을 대입한다.
6. dessert배렬의 요소수는 3개이다. food변수가 확장되였을 때 인용부호는 지우기된다. 3개 요소 apple, pie, ice cream이 있다.

7. 배렬의 첫 요소가 인쇄된다. 변수는 인용되지 않았다면 분리된 단어들로 확장된다.
8. 배렬의 두번째 요소가 인쇄된다.
9. “ice cream”이 인용되였기때문에 한개 단어로 취급한다.
10. dessert배렬에 apple pie와 ice cream을 대입한다. :q는 2중인용부호가 변수를 인용하는 방법으로 변수를 인용하는데 리용될수 있다. 즉 $food:q는 “$food”와 같다.
11. 배렬은 2개 문자렬 apple pie와 ice cream으로 이루어 진다.
12. 배렬의 첫 요소 apple pie가 인쇄된다.
13. 배렬의 둘째 요소 ice cream이 인쇄된다.

:x변경자로 인용하기 만일 메타문자를 포함하는 목록에 배렬과 단어들을 만든다면 :x는 쉘이 변수바꿔넣기를 진행할 때 메타문자를 해석하지 못하게 한다.

실례 13-85

1. **> set things = "*.c a?? file[1-5]**
   > echo $#things
   1
2. **> set newthings = ($things)**
   set: No match.
3. **> set newthings = ($things:x)**
4. **> echo $#newthings**
   3
5. **> echo "$newthings[1] $newthings[2] $newthings[3]”**
   **.c a-?? file[1-5]*
6. **> grep $newthings[2]:q filex**

설명

1. 변수 things에 문자렬을 대입한다. 매 문자렬은 통용기호를 포함하며 변수에서 요소수는 하나 즉 한개 문자렬이다.
2. 문자렬 things로 배렬을 만들려고 할 때 TC쉘은 통용기호문자들을 확장하여 things안에서 파일이름바꿔넣기를 하는데 이때 No match라고 통보한다.
3. :x확장자는 쉘이 things변수에서 통용기호를 확장하지 못하게 한다.
4. 배렬 newthings는 3개 요소로 이루어 진다.
5. 배렬의 요소를 인쇄하려면 그것들은 인용부호안에 넣어야 한다. 그렇지 않으면 쉘은 통용기호를 다시 확장하려고 한다.
6. :q는 마치 변수가 2중인용부호안에 넣어 진것처럼 변수를 인용한다. grep프로그람은 파일 filex에서 패턴 a??를 포함하는 임의의 행들을 인쇄한다.

제 12 절. 내부지령

내부지령은 UNIX실행형지령들과 같이 디스크에 있지 않고 TC쉘의 내부코드부분이며 쉘내부로부터 실행된다. 만일 내부지령이 흐름선의 마지막요소를 제외하고 임의의 요소로 되면 자식쉘에서 실행된다. builtins라고 하는 tcsh지령은 모든 내부지령들을 표시한다.

실례 13-86

1. **builtins**

```
:         @           alias      alloc     bg         bindkey    break
breaksw   builtins    case       cd        chdir      complete   continue
default   dirs        echo       echotc    else       end        endif
endsw     eval        exec       exit      fg         filetest   forreach
glob      goto        hashstat   history   hup        if         job
kill      limit       log        login     logout     ls-F       nice
nohup     notify      onintr     popd      printenv   pushd      rehash
repeat    stop        suspend    switch    telltc     time       umask
unalias   uncomplete  unhash     unlimit   unset      unsetenv   wait
```

내부지령들의 목록을 표 13-24에 주었다.

표 13-24. 내부지령과 의미

내부지령	의 미
:	빈지령을 해석하지만 동작을 진행하지 않는다.
alias [name[wordlist]]	지령에 대한 가명이다. 인수가 없으면 모든 별명들을 인쇄한다. 이름이 있으면 그 별명에 대한 이름을 인쇄하고 이름과 단어목록이 있으면 그 별명을 설정한다.
alloc	요구되는 동적기억기용량을 현시한다. 그것은 사용된 기억기와 나머지기억기로 구분되는데 체계마다 다르다.
bg[%job] %job &	배경에서 현재 또는 지적된 일감들을 실행시킨다. bg내부지령과 같다.
bindkey [-l\ -d\ -e\ -v\ -u] (+) bindkey [-a\ -b\ -k\ -r\ --]key bindkey [-a\ -b\ -k\ -c|-s] [--] key command	추가선택이 없으면 첫번째 형식은 모든 맺기건들과 그것들이 맺어 지는 편집기지령들을 표시하고 두번째 형식은 건이 맺어 지는 편집기지령을 표시하며 세번째 형식은 편집기지령을 건에 결합시킨다. 추가선택은 다음과 같다. -l 모든 편집기지령들과 그것들에 대한 간단한 설명을 표시한다. -d 모든 건들을 기정편집기를 위한 표준결합으로 결합시킨다. -e 모든 건들을 표준GNU의 emacs와 같은 건맺기로 한다. -v 모든 건들을 표준vi와 같은 건맺기로 한다. -a 건맺기들을 대리건배치로 표시하거나 변화시킨다. 이것은 vi지령방식에서 리용되는 건배치이다.

(표계속)

내부지령	의 미
	-b 이 건은 ^문자(실례로 ^A) 또는 C-문자(C-A)로 씌여 진 조종문자, M-문자(M-A)로 씌여 진 메타문자, F-문자렬(F-문자렬)로 씌여 진 기능건 또는 X-문자(X-A)로 씌여 진 확장된 앞붙이건으로 해석된다. -k 이 건은 우아래로, 좌우로 방향이 붙은 방향건이름으로 해석된다. -r 건맺기를 지우기한다. bindkey-r가 건을 자체삽입지령과 맺어 주지 못하고 건을 완전히 해제한다는데 주의하여야 한다. -c 이 지령은 편집지령대신 내부 또는 외부지령으로 해석된다. -s 이 지령은 문자렬로 되고 건이 입력될 때 말단입력으로 취급된다. 지령에서 맺기된 건들은 다시 해석되는데 이것은 10준위해석으로 계속된다. -- 선택처리로부터 벗어 나 다음단어가 련결부호(-)로 시작되면 건으로 되게 한다. -u (혹은 임의의 무효한 추가선택) 사용법통보문을 인쇄한다. 이 건은 단일문자나 문자렬이 될수 있다. 지령이 문자렬과 맺어 지면 그 문자렬의 첫번째 문자는 문자렬안내와 맺어 지고 전체 문자렬은 지령과 맺어진다. 건의 조종문자들은 문자(그것들은 앞에 인용부호안에 있는 편집기지령을 삽입하여 입력될수 있는데 보통 ^V에 맺기된다.)일수 있고 또는 ^-문자형식으로(^A와 같이) 씌여 질수 있다. 지우기는 ^? (^물음표)로 씌여 진다. 건과 지령은 다음과 같이 거꿀빗선, 이 앞에 놓인 탈출문자들을 포함할수 있다(체계 V의 echo형식에서). \a 종 \f 페지바꾸기 \t 수평타브 \b 뒤방향지우기 \n 행바꾸기 \v 수직타브 \e 탈퇴 \r 행복귀 \nnn 8진수 nnn에 대조하는 ASCII문자 \은 그뒤에 오는 문자가 임의의 거꿀빗선(\ )과 탈자기호(^)를 가지고 있으면 그 문자의 특수의미를 없앤다.
break	제일 안쪽 foreach나 while순환에서 탈퇴한다.
breaksw	switch에서 벗어 나 endsw뒤에서 다시 시작한다.
builtins	모든 내부지령들의 이름을 인쇄한다.
bye	logout내부지령과 같다. 쉘이 그렇게 번역될 때에만 리용할수 있다. version쉘변수를 볼것.
case label	switch명령문에서 표식이다.
cd [dir]	쉘작업등록부를 dir로 변경시킨다. 인수가 주어 지지 않으면 사용자의 홈등록부에로 변경시킨다.
cd[-p][-l][-n\ -v][name]	등록부의 이름이 주어 지면 쉘의 작업등록부를 그 이름으로 변경시킨다. 만일 이름이 주어 지지 않으면 홈으로 변경시킨다. 만일 이름이 -이면 이전 작업등록부로 해석된다. -p dirs와 같이 마지막등록부탄창을 인쇄한다.

(표계속)

내부지령	의 미
	-l, -n 그리고 -v기발들은 dirs에 대해서와 같이 cd에 대해서 같은 영향을 준다. 그것들은 -p를 포함한다.
chdir	cd내부지령과 같다.
complete [command [word/pattern/list[:select]/[[suffix]/] ...]]	인수가 없으면 모든 완성들을 표시한다. 지령이 있으면 그 지령에 대한 완성을 표시한다. 지령과 단어 등으로 완성을 정의한다(780페지의 《프로그람작성완성》을 참고).
continue	제일 가까운 while 또는 foreach의 실행을 계속한다.
default:	switch명령문에서 default선택을 표시한다. 기정값은 모든 선택표식다음에 놓여야 한다.
dirs[-l] [-n|-v] dirs-S|-L[filename] dirs-c	첫번째 형식은 등록부탄창을 인쇄한다. 탄창의 꼭대기는 왼쪽에 있고 탄창에 있는 첫번째 등록부는 현재등록부이다. 출력에 있는 -l, ~ 또는 ~name은 홈이나 사용자이름에 대한 홈등록부의 경로이름으로 확장된다. (+) -n 입력은 현시장치에 나타나기전에 묶어 진다. (+) -v 입력은 그것들의 탄창위치들이 앞에 붙은 행단위로 하나씩 인쇄된다. -S 두번째 형식은 등록부탄창을 cd와 pushd지령들의 결합으로 파일이름에 보존한다. -L 보존된 등록부탄창파일로 되는 쉘의 원천파일이름 어느 경우에나 dirsfile은 파일이름이 주어 지지 않으면 사용되고 dirsfile이 해제되면 ~/cshdirs가 사용된다. -c를 가진 형식은 등록부탄창을 지운다.
echo [-n] list	목록에 있는 단어들을 SPACE문자로 분리시켜 쉘의 표준출력에 쓴다. 출력은 -n추가선택이 리용되지 않으면 NEWLINE로 끝난다.
echo [-n] word	매 단어를 쉘의 표준출력으로 쓰는데 그것들은 공백으로 분리되고 행바꾸기로 끝난다. echo_style쉘변수를 기발들과 BSD나 체계 V판본의 echo의 탈출문자들을 지우도록(또는 지우지 않도록) 설정할수 있다.
echotc[-sv]arg	arg에 있는 말단능력을 검사한다. 실례로 echotc home의 유표를 홈위치에 보낸다. 만일 인수가 baud, cols, lines, meta 또는 tabs이면 그 능력값을 인쇄한다. -s를 가지면 존재하지 않는 능력은 오유를 발생시키지 않고 빈 문자렬을 돌려 준다. -v를 리용하면 자세한 통보문이 생긴다.
else if (expr2)then	369페지에서 《if/else if명령문》을 참고

else
end
endif
endsw
	아래에 있는 foreach, if, switch 그리고 while명령문에 대한 설명을 참고한다.

end	while과 그와 맞추어 지는 end사이의 지령들을 expr가 0이 아닌 동안 실행한다. while과 end는 입력행에 단독으로 나타나야 한다. break와 continue는 순환을 끝내거나 계속하도록

(표계속)

내부지령	의 미
	하는데 사용할수 있다. 만일 입력이 말단이면 foreach를 사용할 때와 같이 첫번째 순환때 사용자의 입력을 재촉한다.
eval arg...	인수들을 쉘에 대한 입력으로 취급하고 현재쉘의 환경에서 그 결과 생기는 지령들을 실행한다. 이것은 보통 구문해석이 지령이나 변수바꿔넣기전에 진행되기때문에 그 결과로 생기는 지령들을 실행하는데 사용한다.
eval command	지령을 쉘에 대한 표준입력으로 실행하고 그 결과 생기는 지령들을 실행한다. 이것은 보통 구문해석이 지령이나 변수바꿔넣기전에 진행되므로 그 결과로 생기는 지령들을 실행하는데 리용된다(실례로 eval tset –s 추가선택).
exec command	현재쉘에서 지령을 실행하여 완료한다.
exit [(expr)]	상태변수의 값 또는 expr에 의해 지정된 값을 가지고 쉘에서 탈퇴한다.
fg[%job] %job	현재 또는 지적된 일감을 전경으로 넘긴다. fg내부지령과 같다.
filetest –op file	op(파일요구연산자)를 매 파일에 적용하고 그 결과를 공백으로 분리된 목록으로 되돌린다.
foreach name(wordlist) ...... end	379페지의 《foreach순환》을 참고
foreach var(wordlist)	379페지의 《foreach순환》을 참고
getspath	체계실행경로를 인쇄한다(TCF만)(tcsh에서만).
getxvers	실험적인 판본앞붙이를 인쇄한다(TCF만)(tcsh에서만).
globwordlist	단어목록에서 파일이름확장을 진행한다. echo와 같지만 탈퇴(\ )가 인식되지 않는다. 단어들은 출력에서 빈문자들에 의해 구분된다.
goto label	372페지의 《goto》를 참고
goto word	단어는 파일이름이고 지령대입이 진행되여 항 label물음표로 이행한다. 쉘은 입력을 가능한껏 순환시키고 항 label:(그 앞에 빈 칸이나 타브가 놓일수 있다.)의 행을 탐색하여 그 다음행에서 실행을 계속한다.
hashstat	통계행을 인쇄하여 내부하쉬표가 지령들을 찾는데서 어떻게 사용됐는가를 나타낸다(execx를 피한다.). exec는 hash함수가 가능한 성공을 나타내는 경로의 매 요소에 대해서, 거꿀빗선으로 시작하지 않는 매 요소에서 시도된다.
history [-hTr] [n] history –S|-L|-M [filename] history –c	첫번째 형식은 리력사건목록을 인쇄한다. n이 주어 지면 n개의 제일 최근의 사건들만이 인쇄되거나 기억된다. –h를사용하면 리력목록은 앞에 수들이 없이 인쇄된다. –T가 지적되면 timestamp가 설명문형태로 인쇄된다(이것은 history –L 또는 source –h로 적재하기 편리한 파일들을 생성하는데 사용될수 있다.).

(표계속)

내부지령	의 미
	–r를 사용하면 인쇄순서를 제일 오래된것을 첫번째로 하지 않고 제일 최근의것을 첫번째로 한다(763페지의 《리력》에서 보여 준다.). -c를 리용하면 리력목록을 지운다.
hup [command]	command가 있을 때 중단신호를 받으면 탈퇴하도록 command를 실행하며 쉘이 탈퇴할 때 정지신호를 보내도록 한다. 지령들이 중단신호들에 대해 자체의 응답을 설정하여 hup를 무시할수 있다. 인수가 없으면(쉘스크립트에서만 허용된다.) 쉘이 스크립트의 나머지부분에 대하여 중지신호를 받고 탈퇴하게 한다.
if (expr)command	expr가 참이면 지령이 실행된다. 지령에서 변수바꿔넣기가 먼저 진행되고 동시에 if지령이 나머지부분에 대하여 진행된다. 지령은 단순지령이여야 하고 별명이나 흐름선, 지령목록 또는 괄호속에 있는 지령목록이 되지 말아야 한다. 그러나 그것은 인수들을 가질수 있다. 입출력방향바꾸기는 expr가 거짓일 때에도 진행되며 이때 지령은 실행되지 않는다. 즉 이것은 오유이다.

if (expr) then
...
else if (expr2) then
...
else
...
endif
	지적된 expr가 참이면 첫번째 else에 대한 지령을 실행한다. 그렇지 않고 expr들이 참이면 두번째 else에 대한 지령들이 실행한다. else if쌍의 개수는 임의로 할수 있다. 다만 한개의 endif가 요구된다. else부분은 선택적이다(단어 else와 endif는 입력행의 시작전에 나타나야 한다. if는 입력행우에 홀로 나타나거나 else뒤에 나타나야 한다.).

inlib shared-library...	매 shared-library를 현재환경에 추가한다. 공유된 서고를 지우기하는 방법은 없다(Domain/OS에서만 가능하다.).
jobs[-l]	일감조종하에서 능동일감들을 표시한다. –l이 있으면 표준정보외에 ID들을 표시한다.
kill[-sig] [pid] [%job]... kill-l	TERM(끝)신호를 기정으로 또는 지정된 신호로 지적된 ID, 표시된 일감 또는 현재일감에 보낸다. 신호들은 번호나 이름에 의해 주어 진다. 기정값은 없다. kill을 입력하면 신호를 현재일감에게 보내지 않는다. 보낸 신호가 TERM(끝) 또는 HUP(정지)이면 일감이나 프로쎄스에 CONT(계속)신호도 보낸다. –l이 있으면 전송할수 있는 신호이름들을 표시한다.
limit [-h] [resource[max-use]]	현재프로쎄스나 그것이 파생시킨 임의의 프로쎄스에 의한 자원소비를 제한하는데 지적된 자원에서 max-use를 초과하지 못한다. max-use를 생략하면 현재한계를 인쇄한다. 자원을 생략하면 모든 한계들을 현시한다. –h가 있으면 현재한계대신 장치적한계를 사용한다. 주사용자만이 장치적한계를 올릴수 있다. 자원은 프로쎄스당 최대 CPU시간(초)인 cputime, 허용되는 단일파일의 최대크기인 filesize, 프로쎄스에 대한 최대자료크기(탄창포함)인 datasize, 프로쎄스에 대한 최대탄창크기인 stacksize, core dump의 최대크기인 coredump 그리고 파일형식자에 대한 최대값 descriptors들중 어느 하나가 될수 있다.
log	쉘변수 watch를 인쇄하고 언제 마지막으로 등록가입했든지 관계없이 watch에 표시된 매 사용자에게 누가 등록가입되는가를 통보한다. watch log를 볼것.

(표계속)

내부지령	의 미
login	등록가입쉘을 완료하여 그것을 /bin/login의 실체로 교체한다.
login [username|-p]	등록가입쉘을 완료하고 등록가입을 호출한다. .logout파일은 처리하지 않는다. 사용자이름을 생략하면 login이 사용자의 이름을 재촉한다. –p가 있으면 현재환경(변수들)을 보존한다.
logout	등록가입쉘을 완료한다.
ls-F [-switch...] [file...]	ls-F와 같이 파일들을 표시하지만 더 빠르다. 그것은 특수문자로 표시된 특수파일의 형태를 식별한다. / 등록부 * 실행형 # 블로크장치 % 문자장치 | 이름 지어 진 파이프(이름 지어 진 파이프를 가진 체계들에 대해서만) = 소케트(소케트를 가진 체계들만) @ 기호적련결(기호적련결을 가지고 있는 체계들만) + 숨겨 진 등록부(AIX에서만) 또는 상황의존형(HP/UX에서만) : 망특수기호(HP/UX에서만) 만일 listlinks쉘변수를 설정하면 기호적련결이 더 상세하게 식별된다(물론 그것들을 가지고 있는 체계들에서만 가능하다.). @ 등록부가 아닌것에 대한 기호적련결 > 등록부에 대한 기호적련결 & 아무데도 기호적련결을 하지 않는다. ls-F내부지령은 파일형태나 확장자에 의존하는 여러가지 색갈들을 리용하여 파일들을 표시할수 있다.
migrate [-site] pid|%jobid...	첫번째 형식은 프로쎄스나 일감을 지적된 싸이트나 체계경로에 의해 결정되는 기정싸이트에로 보낸다. 두번째 형식은 migrate-site$$와 같다. 그것은 현재프로쎄스를 지적된 싸이트로 보낸다. 쉘자체를 보내면 쉘이 자기의 tty를 잃으려고 하지 않으므로 예상치 않은 방식의 동작을 할수 있다(TCF에서만).
@ @ name=expr @ name[index]=expr @ name++|- - @ name[index]++|- -	첫번째 형식은 모든 쉘변수들의 값을 인쇄한다. 두번째 형식은 expr의 값을 이름에 대입한다. 세번째 형식은 expr의 값을 이름의 첨수번호요소에 대입한다. 이름과 그의 첨수번호요소는 둘다 미리 존재해야 한다. 네번째와 다섯번째 형식은 이름이나 그의 첨부번호요소를 증가(++) 또는 감소(- -)시킨다.
newgrp [-] group	exec newgrep와 같다. 쉘이 그렇게 번역되였을 때만 리용할수 있다. version쉘변수에서 보여 준다.
nice [+number] [command]	쉘에 대한 일정순서작성우선권을 수로 설정한다. 수가 없으면 4로 설정한다. 지령이 있으면 적당한 순서로 실행시킨다. 수가 클수록 프로쎄스는 CPU를 덜 리용한다. 주사용자는 nice-number...를 리용하여 부수의 우선권을 지정할수 있다. 지령은 언제나 자식쉘에서 실행되고 명령문이 적응되면 지령에 대한 제한은 간단하다.

(표계속)

내부지령	의 미
nohup [command]	다음 HUP(중지)들을 무시하여 지령을 실행한다. 인수가 없으면 스크립트의 나머지부분에서 HUP부분을 무시한다.
notify [%job]	현재 또는 지적된 일감의 상태가 변할 때 사용자에게 비동기적으로 알려 준다.
onintr [-|label]	새치기상태에서 쉘의 동작을 조종한다. 인수가 없으면 onintr는 새치기상태에서 쉘의 기정동작을 회복한다(쉘은 쉘스크립트들을 끝내고 말단지령입력준위로 되돌아 간다.). 덜기기호인수가 있으면 쉘은 모든 새치기들을 무시한다. label인수가 있으면 쉘은 새치기를 접수할 때 goto label을 실행하거나 자식프로쎄스가 새치기되였기때문에 완료된다.
popd [+n]	등록부탄창을 뽑아서 새로운 꼭대기등록부로 변경한다. 등록부탄창의 요소들이 꼭대기에서 시작하여 0부터 번호가 붙는다. +n이 있으면 n번째 탄창의 입력을 버린다.
printenv [name]	다음 모든 환경변수들의 이름과 값들을 인쇄한다. 이름이 있으면 환경변수이름의 값을 인쇄한다.
pushd [+n | dir]	등록부를 등록부탄창에 넣는다. 인수가 없으면 꼭대기 2개의 요소들을 교환한다. +n이 있으면 n번째 입력을 등록부의 꼭대기에 놓고 거기로 등록부를 변화시킨다. Dir가 있으면 현재작업등록부를 탄창에 넣어 dir로 변화시킨다.
rehash	추가된 새로운 지령들을 고려하여 path변수에 표시된 등록부내용들로 된 내부하쉬표를 다시 계산한다.
repeat count command	지령을 count만큼 반복한다.
rootnode//nodename	rootnode를 //nodename으로 변경하여 /이 //nodename으로 해석된다(Domain/OS에서만 가능하다.).
sched sched [+]hh:mm command sched-n	첫번째 형식은 일정순서를 작성한 사건리력을 인쇄한다. sched 쉘변수는 일정순서를 작성한 사건목록을 인쇄하는 형식을 정의하기 위해 설정할수 있다. 두번째 형식은 지령을 일정순서를 작성한 사건목록에 추가한다.
set set name... set name=word... set [-r] [-f\ -l] name=(wordlist)...(+) set name[index]=word... set-r set-r name... set-r name=word...	지령의 첫번째 형식은 쉘변수들의 값을 인쇄한다. 한개이상의 단어를 포함하는 변수들은 괄호안에 있는 단어목록으로 인쇄한다. 두번째 형식은 이름을 빈 문자렬로 설정한다. 세번째 형식은 이름을 단일문자로 설정한다. 네번째 형식은 이름을 단어목록에 있는 단어들의 목록으로 설정한다. 모든 경우에 변수는 확장된 지령이나 파일이름이다. –r가 지정되면 값은 읽기전용으로 설정된다. –f 또는 –l이 지적되면 그것들의 순서를 유지하는 유일한 단어들만을 설정한다. –f추가선택은 단어의 첫번째를, -l은 단어의 마지막을 설정한다. 다섯번째 형식은 이름의 첨수번째 요소를 단어로 설정한다. 이 요소는 미리 존재해야 한다. 여섯번째 형식은 읽기전용인 모든 변수들의 이름을 표시한다. 일곱번째 형식은 이름이 값을 가지든 가지지 않든 그것을 읽기전용으로 만든다(tcsh에서만). 여덟번째 형식은 세번째 형식과 같지만 이름을 읽기전용으로 만든다(tcsh에서만).

(표계속)

내부지령	의 미
set [var[=value]]	809페지의 《변수》를 참고
setenv [VAR[word]	809페지의 《변수》를 참고한다. 가장 일반적으로 사용되는 환경변수들인 USER, TERM 그리고 PATH는 자동적으로 csh변수들인 user, term, path에로 반출하거나 그로부터 반입할수 있다. 이를 위해 setenv를 리용할 필요는 없다. 이외에도 쉘은 csh변수인 cwd가 변할 때마다 그로부터 환경변수 PWD를 설정한다.
setenv [name[value]]	인수가 없이 모든 환경변수들의 이름과 값들을 인쇄한다. 이름이 주어 지면 환경변수의 이름을 값으로 설정하고 값이 없으면 빈 문자렬로 설정한다.
setpath path	setpath와 같다(Mach에서만).
setpath LOCAL|site|cpu	체계실행경로를 설정한다.
settc cap value	말단능력 cap가 값 value를 가지고 있다는것을 쉘에 다시 알려준다. 정확한 검사가 진행되지 않는다. 말단사용자들은 제일 오른쪽 끌렬에서 정확히 끝내도록 settc xn no를 사용할수 있다.
setty [-d\ -q\ -x] [-a] [[+\ -]mode]	쉘이 어느 tty방식을 변경시키지 않도록 하는가를 조종한다. D, -q 또는 –x는 setty가 tty방식의 edit quote 또는 execute모임에서 각각 동작하도록 알려 준다. D, -q 또는 –e가 없으면 execute가 실행된다.다른 인수들이 없으면 setty는 설정된 (+mode) 또는 설정되지 않은(-mode) 선택된 모임에 있는 방식들을 표시한다. 가능한 방식들과 현시는 체계에 따라 다르다. 실례로 setty+echok echoe는 echok방식을 설정하여 지령들이 echo방식을 설정하거나 해제하게 한다. 이 경우 쉘은 지령들을 실행한다.
setxvers [string]	문자렬에 대한 실험적판본앞붙이를 설정하거나 문자렬을 생략하면 그것을 지우기한다(TCF에서만).
shift[variable]	argv 또는 variable의 요소들이 주어 지면 왼쪽으로 자리밀기되여 첫번째 요소를 없앤다. 변수가 설정되지 않거나 빈 값을 가지면 오유로 된다.
source [-h]	이름으로부터 지령들을 읽는다. source지령들은 겹싸여 질수 있지만 너무 깊게 겹싸여 지면 쉘이 파일형식자에서 벗어 날수 있다. 임의의 준위에서 원천화된 파일오유는 모든 겹싸여 진 source지령들을 끝낸다. 일반적으로 현재쉘에서 처리되는 변수설정들을 담보하기 위해 .login 또는 .cshrc파일들을 재실행시키는데 리용된다. 즉 쉘은 자식쉘을 작성하지 않는다. –h추가선택이 있으면 지령들을 실행하지 않고 리력목록에 있는 파일이름으로부터 그것들을 넣는다.
stop[%job]...	현재 또는 지정된 배경일감을 정지한다.
suspend	쉘은 ^Z인 정지신호를 보낸것처럼 쉘을 그 경로에서 정지시킨다. 이것은 su에 의해 시작된 쉘들을 정지시키는데 자주 리용된다.
switch[string]	376페지의 《switch지령》을 볼것.
telltc	모든 말단능력들의 값들을 표시한다.

(표계속)

내부지령	의 미
time [command]	인수가 없으면 이 쉘과 그 자식쉘에 의해 리용된 총시간을 인쇄한다. 추가선택으로 command를 선택하면 command를 실행하고 그것이 리용한 총시간을 인쇄한다.
umask [value]	파일작성마스크를 현시한다. 값이 있으면 파일작성마스크를 설정한다. 파일들에 대한 666허가와 등록부에 대한 777허가의 배타적론리합으로 8진수값이 주어 지면 새로운 파일에 대한 허가를 얻는다. 허가는 umask를 사용하여 추가할수 없다.
unalias pattern	이름이 패턴과 일치되는 모든 별명들을 지우기 한다. 따라서 unelias*는 모든 별명들을 지운다. 지워 지는 별명이 없어도 오유로 되지 않는다.
uncomplete pattern	그 이름이 패턴과 대조하는 모든 완성들을 지운다. 따라서 uncompete*는 모든 완성들을 지운다.
unhash	내부하쉬표를 불가능하게 한다.
universe universe	universe를 universe로 설정한다(Masscomp/RTU에서만).
unlimit [-h] [resource]	자원에 대한 한계를 지운다. 한계가 지적되지 않으면 모든 자원한계를 지운다. –h가 있으면 대조하는 장치적한계를 지운다. 주사용자만이 이것을 할수 있다.
unsetenv pattern	이름이 패턴과 대조하는 모든 환경변수들을 지우기 한다. 따라서 unsetenv*는 모든 환경변수들을 지우기 한다. 이렇게 하는 것은 좋은 방법이 아니다. 지우기되는 패턴이 없어도 이 내부지령으로부터 오유가 생기지 않는다.
unsetenv variable	환경으로부터 변수를 삭제한다. unset를 리용할 때와 같이 패턴대조가 진행되지 않는다.
@ [var=expr] @ [var[n]=expr]	인수가 없으면 모든 쉘변수들에 대한 값을 현시한다. 인수가 있으면 변수 var 또는 var의 값에 있는 n번째 단어가 expr가 지정하는 값으로 설정된다.
ver [systype[command]]	인수가 없으면 SYSTYPE를 인쇄한다. systype가 있으면 SYSTYPE를 systype로 설정한다. systype와 command가 있으면 systype에서 command를 실행한다. systype는 bsd4.3 혹은 sys5.3으로 될수 있다(Domain/OS에서만).
wait	재촉문을 현시하기전에 배경과제가 끝나기(또는 새치기)를 기다린다.
warp universe	universe를 universe로 설정한다(Convex/OS에서만).
watchlog	내부지령 log에 대한 또 다른 이름이다. 쉘이 그렇게 번역되였을 때에만 리용할수 있다. version쉘변수를 볼것.
where command	경로에 알려 진 별명들, 내부지령들 그리고 실행형들의 모든 실체를 알려 준다.
which command	바꿔넣기, 경로탐색 등 후에 쉘에 의해 실행되는 지령을 현시한다. 내부지령은 which와 같지만 그것은 tcsh별명들과 내부지령들을 정확히 알려 주며 그 속도는 10~100배 더 빠르다.
while(expr)	382페지의 《while순환》을 참고

1. 특수한 별명

TC쉘의 모든 별명들은 설정되면 지적된 시간에 자동적으로 실행한다. 그것들은 모두 초기에 정의되지 않는다.

beepcmd	쉘이 말단종을 울리려고 할 때 실행된다.
cwdcmd	작업등록부가 변할 때마다 실행된다.
periodic	teriod분마다 실행된다. 실례로 >set tperiod=30 alias period date
precmd	재촉문이 인쇄되기전마다 실행된다. 실례로 alias precmd date

2. 특수내부쉘변수

내부쉘변수들은 쉘에 대해 특수한 의미를 가지고 있으며 많은 쉘지령들이 동작하는 방식을 수정하고 조종하는데 사용된다. 쉘내부변수들은 국부변수들이며 따라서 쉘내부변수들의 대부분이 자식 T C쉘에 넘겨 져 영향을 주면 .tcshrc파일에서 설정된다.

쉘이 시작할 때 그것은 자동적으로 다음변수들 즉 addsuffix, argv, autologout, command, echo_style, edit, gid, group, home, loginsh, oid, path, prompt, prompt2, prompt3, shell, shlvl, tcsh, term, tty, uid, user와 version을 설정한다. 사용자가 그것들을 변화시키려고 하지 않으면 이 변수들은 고정된다. 쉘은 cwd, dirsturck, owd, status와 같이 주기적인 갱신을 필요로 하는 특수변수들을 변화시키고 그 경로를 유지하며 사용자가 등록탈퇴할 때 쉘은 logout변수를 설정한다. 일부 국부변수들은 같은 이름을 대조하는 환경변수를 가지고 있다. 만일 환경변수나 국부변수들의 어느 하나가 사용자의 환경에서 변화되면 쉘은 국부변수와 환경변수[10]들을 다같이 변화시켜 그 값들이 언제나 대조되게 한다.

서로 대조되는 변수들의 실례는 afuser, group, home, path, shlvl, term, user 등이다(비록 cwd와 pwd가 같은 의미를 가지지만 그것들은 서로 대조되지 않는다. 지어 path와 PATH변수들에 대한 문법은 다르지만 그것들은 어느 하나가 변경된다면 자동적으로 서로 대조된다.).

표 13-25. 특수쉘변수*

변 수	기 능
addsufix	파일이름완성을 위해 등록부의 끝에 빗선을 추가하고 보통 파일들이 대조되면 그 끝에 공백을 추가한다. 기정으로 설정된다.
afsuser	설정되면 autologout의 자동열쇠특성은 kerberos인증을 위해 국부사용자이름대신 그 값을 사용한다.
ampm	설정되면 언제나 12시간 AM/PM형식으로 표시된다.
argv	쉘에 대한 지령행인수들의 배렬이다. $1, $2 등과 같이 표시된다.

10. 이것은 변수가 읽기전용변수가 아닌 경우에만 참이고 동기가 없다.

(표계속)

변 수	기 능
autocorrect	파일이름, 지령 또는 변수완성을 하기전에 맞춤법검사기를 호출한다.
autoexpand	설정되면 확장리력편집기지령은 완성이 매번 진행되기전에 자동적으로 호출된다.
auotlist	설정되면 모호한 완성후에 가능성을 표시한다. ambigous로 설정되면 완성에 의해 새 문자들이 추가되지 않을 때만 가능성들을 표시한다.
autologout	자동등록탈퇴전에 인수(분)만큼 비활동상태에 있다. 두번째로 선택할수 있는 인수는 자동열쇠걸기가 현시장치에 열쇠를 걸기전에 비활동상태의 시간(분)이다.
backslash-quote	설정되면 거꿀빗선은 그자체, 단일인용부호 또는 2중인용부호를 인용한다.
cdpath	보조등록부들을 현재등록부에서 찾지 못하면 cd가 그것들을 탐색하는 등록부들의 목록이다.
color	내부지령, ls-F에 대한 색현시를 가능하게 하고 - -color=auto를 ls로 넘긴다.
complete	enhance로 설정되면 완성은 선택을 무시하고 점, 련결부호 그리고 밑선을 단어분리기로 하며 련결부호와 밑선은 같은것으로 된다.
correct	cmd로 설정되면 지령들은 자동적으로 맞춤법된다. 만일 complete로 설정되면 지령들은 자동적으로 완성된다. all로 설정되면 전체 지령행이 정정된다.
cwd	현재작업등록부의 완전경로이름이다.
dextract	설정되면 pushd+n은 n번째 등록부를 꼭대기에서 돌리지 않고 등록부탄창으로부터 뽑아 낸다.
dirsfile	dirs-S와 dirs-L이 리력파일을 찾는 기정위치이다. 설정되지 않으면 ~/.cshdirs를 리용한다.
dirstack	등록부탄창에 있는 모든 등록부들의 목록이다.
dunique	pushd가 탄창에 등록부입력들을 복사해 넣지 않게 한다.
echo	설정되면 인수를 가지는 매 지령은 실행되기전에 출력된다. –x지령행추가선택에 의해 설정된다.
echo-style	echo에 대한 형식을 설정한다. bsd로 설정되면 첫번째 인수가 –n일 때 행바꾸기문자를 출력하지 않는다. sysv로 설정되면 출력문자렬에서 거꿀빗선이 붙은 탈출문자들을 인식한다. both로 설정되면 –n기발과 거꿀빗선이 있는 탈출문자들을 인식한다. 이것은 기정이다. 만일 none으로 설정되면 아무것도 인식하지 않는다.
edit	대화형쉘을 위하여 지령행편집기를 설정한다. 기정으로 설정된다.
ellipsis	설정되면 %c/%.과 %C재촉문문자렬(756페지에서 《쉘재촉문》을 참고)이 /<skipped>대신 ellipsis(...)로 빠진 등록부들을 표시한다.
fignore	완성에 의해 무시되는 파일이름뒤붙이들을 표시한다.
filec	tcsh에서는 완성이 언제나 리용되며 이 변수는 무시한다. csh에서 설정되면 파일이름완성을 진행한다.
gid	사용자읽기그룹 ID번호
group	사용자그룹이름

(표계속)

변 수	기 능
histchars	리력바꿔넣기에서 리용되는 문자들을 결정하는 문자렬값이다. 값의 첫번째 문자는 기정문자인 !를 대신하는 리력바꿔넣기문자로 사용한다. 값의 두번째 문자는 속성바꿔넣기에서 문자 ^를 대신한다.
histdup	리력목록에 있는 사본입력들의 처리를 조종한다. all(모든 사본들을 지우기한다.), prev(이전의 지령을 복사하면 현재지령을 지우기한다.) 또는 erase(보다 이전의 복사한 사건대신 현재사건을 삽입한다.)로 설정될수 있다.
histfile	history-S와 history-M이 리력파일을 찾는 기정위치이다. 설정되지 않으면 ~/.history가 리용된다.
histlit	리력목록에 사건들을 문자그대로 입력한다. 즉 리력바꿔넣기에 의해 확장되지 않는다.
history	첫번째 단어는 기억될 리력사건들의 개수를 표시한다. 선택할수 있는 두번째 단어(+)는 리력이 인쇄되는 형식을 표시한다.
home	~와 같은 사용자의 홈등록부이다.
ignoreeof	Control-D를 눌러서 등록탈퇴하면 Use exit를 인쇄하고 tcsh를 벗어 난다. 부주의로 등록탈퇴하지 않게 한다.
implicitcd	설정되면 쉘은 지령행에 입력된 등록부이름을 그것이 마치도 그 등록부에로 변경시키기 위한것으로 처리하고 변화시킨다.
inputmode	insert 또는 overwrite로 설정되면 편집기를 매행의 시작점에서 입력방식으로 넣는다.
listflags	x, a, A 또는 그 조합(xA)으로 설정되면 값들이 ls-F에 대한 기발로 리용되여 ls-xF, ls-Fa, ls-FA 또는 이 기발들의 임의의 조합처럼 동작하게 한다.
listjobs	설정되면 일감을 중단할 때 모든 일감들을 표시한다. long으로 설정되면 긴 형식으로 표시된다.
listlinks	설정되면 ls-F내부지령은 매 기호련결이 지정하는 파일의 형태를 보여 준다.
listmax	list-choices편집기지령이 처음에 질문이 없이 표시하는 항목들의 최대수이다.
listmaxlows	list-choices지령이 처음에 질문이 없이 표시하는 항목들의 최대행수이다.
loginsh	쉘이 등록가입쉘이면 그 쉘에 의해 설정된다. 쉘안에서 그것을 설정하거나 해제해도 아무런 영향도 미치지 않는다. 이 표에서 shlvl을 볼것.
logout	쉘에 의해서 정상등록탈퇴하기전에 normal로 설정되고 자동탈퇴하기전에 automatic로 설정되며 쉘이 중지신호에 의해 사멸되였으면 hangup로 설정된다.
mail	들어 오는 우편들을 검사하는 파일이나 등록부들의 이름이다. 새로운 우편이 들어 온후 10분 지나서 You have new mail을 인쇄한다.
matchbeep	never로 설정되면 완성은 경고음을 울리지 않는다. nomatch로 설정되면 대조되는것이 없을 때에만 경고음을 울린다. ambgious로 설정되면 대조가 여러개일 때 경고음을 울린다.
nobeep	모든 경고음이 울리지 않게 한다.

(표계속)

변 수	기 능
noclobber	방향바꾸기가 리용될 때 현존파일이 부주의로 지워 지지 않게 한다. 실례로 ls>file
noglob	설정되면 통용기호를 리용할 때 파일이름과 등록부탄창바꿔넣기를 하지 못하게 한다.
nokanji	설정되고 쉘이 Kanji(version쉘변수에서 보여 준다.)를 지원하면 Meta건이 리용되도록 그것을 쓸수 없게 한다.
nonomatch	설정되면 임의의 현존파일과 대조되지 않는 파일이름바꿔넣기나 등록부탄창바꿔넣기가 오유를 발생하지 않고 그대로 남겨 진다.
nostat	설정되면 쉘은 재촉문이 나타나기전까지 기다리지 않고 비동기적으로 일감완성을 시작한다.
notify	완성조작기간에 종합하지 않는 등록부들의 목록이다. 이것은 보통 종합하는데 많은 시간이 드는 등록부들을 지우기하는데 리용된다.
oid	사용자의 실지 조직 ID이다(Domain/OS에서만).
owd	낡은 또는 이전의 작업등록부이다.
path	실행형지령들을 찾는 등록부들의 목록이다. PATH환경변수로부터 기동할 때 path가 쉘에 의해 설정되고 또는 PATH가 존재하지 않으면 /usr/local/bin/usr/bsd/bin/usr/bin과 같은 체계종속적인 기정값으로 설정된다.
printexitvalue	설정되고 대화형프로그람이 0이 아닌 상태로 탈퇴하면 쉘은 탈퇴상태를 인쇄한다.
prompt	말단으로부터 매 지령행을 읽어 들이기전에 인쇄되는 문자렬이다. 특수형식문자렬들을 포함할수 있다(756페지의 《쉘재촉문》을 참고).
prompt2	while과 for순환에서 그리고 \ 으로 끝나는 행뒤에 나타나는 문자렬이다. prompt에서와 같은 형식문자렬들이 리용될수 있다. %R의 변수의미를 주목해야 한다. 대화형쉘에서 기정으로 %R?로 설정된다.
prompt3	자동맞춤법을 진행할 때 나타나는 문자렬이다. prompt에서와 같은 형식문자렬들이 리용될수 있다. %R의 변수의미를 주목해야 한다. 대화형쉘에서 기정으로 CORRECT>%R(y|n|e|a)?로 설정된다.
promptchars	설정되면(2개 물음표로 된 문자렬) 콤퓨터에서 쉘변수가 보통사용자에 대한 첫번째 문자와 주사용자에 대한 두번째 문자로 교체되는 %#형식화문자렬이다.
pushtohome	설정되면 인수가 없는 pushd는 cd와 같이 pushd~와 같다.
pushdsilent	설정되면 pushd와 popd는 등록부탄창을 인쇄하지 않는다.
recexact	설정되면 보다 더 긴 대조를 할수 있을 때에도 완성은 정확한 대조에 대해서만 진행된다.
recognize_only_ executables	지령표시는 설정되면 실행할수 있는 경로에 있는 파일들만을 인쇄한다.
rmstar	설정되면 rm*가 실행되기전에 사용자에게 입력을 재촉한다.

(표계속)

변 수	기 능
rprompt	화면의 오른쪽에 인쇄하는 문자렬(지령입력후에)인데 이때 재촉문은 왼쪽에 현시되게 된다. 그것은 재촉문과 같은 형식화문자를 인식한다. 또한 그것은 자동적으로 없어 지며 필요할 때 다시 나타나 지령입력이 없어 지지 않는다는것을 보여 주며 재촉문, 지령입력 그리고 그자체가 첫번째 행에 다같이 놓일 때 나타난다. 편집이 설정되지 않으면 rprompt는 재촉문다음에 그리고 지령입력전에 인쇄된다.
savedirs	설정되면 쉘은 탈퇴하기전에 dirs-S를 수행한다.
savehist	설정되면 쉘은 탈퇴하기전에 history-S를 수행한다. 첫번째 단어가 수자로 설정되면 많은 행들이 기억된다(번호는 리력보다 작거나 같아야 한다.). 만일 두번째 단어가 merge로 설정되면 리력목록은 그것을 교체하지 않고 현존리력파일에 합쳐 지며 timestamp로 분류되고 제일 최근의 사건들이 남는다.
sched	sched내부지령이 순서짜기된 사건들을 인쇄하는 형식이다. 주어 지지 않으면 %h\ t%T\ t%R\ n이 리용된다. 형식문자렬은 재촉문아래에 서술된다. %R의 변수의미를 주목하여야 한다.
shell	쉘이 존재하는 파일이다. 이것은 실행비트모임을 가지고 있는 파일들을 해석하기 위하여 쉘들을 파생시키는데 사용되지만 체계에 의해 실행할수 없다(내부지령과 외부지령의 실행에 대한 설명을 볼것.). 쉘(체계의존)의 홈에 초기화된다.
shlvl	겹싸여 진 쉘의 개수이다. 등록가입쉘에서 1로 재설정된다. loginsh를 볼것.
status	마지막지령에 의해서 귀환된 상태이다. 지령이 비정상적으로 완료되면 0200이 상태에 추가된다. 실패한 내부지령들은 1탈퇴상태를 되돌리고 성공한 모든 내부지령들은 0탈퇴상태를 되돌린다.
symlinks	기호련결(symlink)결과를 조종하기 위하여 여러개의 다른 값으로 설정될수 있다(tcsh의 man page를 참고할것.).
tcsh	형식 R.VV.PP에서 쉘의 판본번호인데 여기서 R는 기본계렬번호이고 VV는 현재판본 그리고 PP는 림시수정준위이다.
term	말단형태이다. 749페지의 《기동》에서 서술한바와 같이 보통 ~/.login에 설정된다.
time	수자로 설정되면 time내부지령은 CPU시간보다 많은 시간을 가지는 모든 지령다음에서 자동적으로 실행된다. 두번째 단어가 있으면 그것은 time내부지령의 출력을 위한 형식문자렬로 리용된다. 다음물음표들은 형식문자렬에서 리용될수 있다. %U　CPU시간으로 사용자방식에서 프로쎄스가 소비한 시간 %S　CPU시간으로 핵심부방식에서 프로쎄스가 소비한 시간 %E　초로 리용된 시간 %P　(%U+%S)/%E로 계산된 CPU백분률 %W　프로쎄스가 교환된 반복수 %X　사용된 본문공간의 평균량(KB) %D　사용된 자료/탄창공간의 평균량(KB) %K　사용된 전체 공간(%X+%D) (KB) %M　프로쎄스가 임의의 시간에 리용하는 최대기억기(KB)

(표계속)

변 수	기 능
	%F 기본페지결합의 수(디스크로부터 가져 올 필요가 있는 페지) %R 작은 페지결합의 수 %I 입력조작의 수 %O 출력조작의 수 %r 접수된 소케트통보문의 수 %s 보낸 소케트통보문의 수 %k 접수된 신호들의 수 %w 선택적인 상태추가선택들의 수 %c 비선택적인 상태추가선택들의 수
	첫번째 4개 물음표들만이 BSD자원한계기능들이 없이 체계들에서 지원된다. 기정시간형식은 자원사용법통보를 지원하는 체계들에 대해서는 %Uu %Ss %E %P %X+%Dk %I+%Oio %Fpf+%Ww이고 지원하지 않는 체계들에 대해서는 %Uu %Ss %E %P이다.
	sequent의 DYNIX/ptx에서 %X, %D, %K, %r와 %s는 리용할수 없지만 다음의 추가적인 문자렬들은 리용할수 있다. %Y 진행된 체계호출의 수 %Z 0이 채워 진 페지들의 수 %I 프로쎄스가 차지하는 령역의 크기가 핵심부에 의해 증가된 회수 %d 프로쎄스가 차지하는 령역의 크기가 핵심부에 의해 감소된 회수 %l 진행된 읽기체계호출의 수 %m 진행된 쓰기체계호출의 수 %p 원시디스크구동기로부터 읽은 회수 %q 원시디스크에 쓴 회수
	기정시간형식은 %Uu %Ss $E %P %I+%Oio %Fpf+%Ww이다. CPU백분률은 다중처리기에서 100%보다 더 높을수 있다.
tperiod	periodic특수별명들이 실행되는 사이의 기간(분)
tty	tty의 이름 또는 그에 붙은것이 없을 때 빈 값이다.
uid	사용자의 실지 사용자 ID
user	사용자의 등록가입이름
verbose	설정되면 매 지령의 단어들이 리력바꿔넣기다음에 인쇄되게 한다. –v추가선택우에 설정된다.
version	판본ID를 붙인다. 그것은 쉘의 판본번호, 원본, 계렬날자, 판매자, 조작체계 등을 포함한다.

*. tcsh사용페지에 맞게 하였다.

쉘지령행추가선택 T C쉘은 자기 동작을 조종하거나 수정하기 위한 많은 지령행추가선택(기발인수라고도 하는)을 가질수 있다. 지령행추가선택을 표 13-26에 주었다.

표 13-26. 쉘지령행추가선택

추가선택	의 미
-	쉘이 등록가입쉘이라는것을 지적한다.
-b	추가선택처리로부터 탈퇴하게 한다. 그후에 임의의 쉘인수들은 추가선택들로 취급되지 않는다. 남아 있는 인수들은 쉘추가선택들로 해석되지 않는다. 쉘이 사용자설정 ID이면 이 추가선택을 포함해야 한다.
-c	단일인수가 –C다음에 놓이면 지령들은 그 인수(파일이름)로부터 읽어 진다. 남아 있는 인수들은 argv쉘변수에 놓인다.
-d	쉘은 ~/.cshdirs로부터 등록부탄창을 적재한다.
-Dname[=value]	환경변수이름을 값으로 설정한다.
-e	임의의 호출된 지령이 비정상적으로 끝나거나 0이 아닌 탈퇴상태를 되돌릴 때 쉘이 탈퇴한다.
-f	새로운 TC쉘을 기동할 때 쉘이 ~/.tcshrc를 무시하기때문에 빠른 시동이라고 한다.
-F	vfork대신 fork를 리용하여 쉘은 프로쎄스를 파생시킨다(Convex/OS에서만).
-i	쉘은 대화형이고 그것이 말단이 아닐 때에도 입력을 재촉한다. 이 선택은 입력과 출력이 말단에 련결될 때 필요하지 않다.
-l	-l이 다만 지적된 기발이면 쉘은 등록가입쉘이다.
-m	쉘은 유력한 사용자에게 속해 있지 않으면 ~/.tcshrc를 적재한다.
-n	스크립트를 오유수정하는데 리용된다. 쉘은 지령행을 구문해석하지만 실행하지는 않는다.
-q	쉘은 SIGQUIT신호를 받아서 오유수정기안에서 리용될 때 동작한다. 일감조종을 할수 없다.
-s	지령입력은 표준입력로부터 얻어 진다.
-t	쉘은 한개 행의 입력을 읽어서 실행한다. \ 을 사용하여 이 행의 마지막에서 행바꾸기에로 탈퇴시켜 다른 행에서 계속하게 하는데 리용된다.
-v	지령입력이 리력바꿔넣기후에 출력되도록 verbose 쉘변수를 설정한다. 쉘스크립트를 오유수정하는데 리용한다.
-x	지령이 실행전에 표시되고 리력과 변수바꿔넣기후에 출력되도록 echo쉘변수를 설정한다. 쉘스크립트를 오유수정하는데 리용한다.
-V	~/.tcshrc파일을 실행하기전에 verbose쉘변수를 설정한다.
-X	~/.tcshrc파일을 실행하기전에 echo쉘변수를 설정한다.

*. tcsh사용지도서페지에 맞게 하였다.

ＴＣ쉘 련습

련습문제 57: 기동

1. init프로쎄스는 무엇을 하는가?
2. login프로쎄스의 기능은 무엇인가?
3. 어느 쉘을 리용하고 있는지 어떻게 아는가?
4. 등록가입쉘을 어떻게 변경할수 있는가?
5. .tcshrc, .cshrc, .login파일사이의 차이점을 설명하시오. 어느것이 먼저 실행되는가?
6. .tcshrc파일을 다음과 같이 편집하시오.
   ㄱ. 자체의 별명을 3개 만드시오.
   ㄴ. 재촉문을 주콤퓨터이름, 시간, 사용자이름으로 재설정하시오.
   ㄷ. 다음변수를 설정하고 매 변수뒤에 그것이 무엇을 하는가를 설명하는 설명문을 넣으시오.

7. 다음과 같이 입력하시오.
   source .tcshrc
   source지령은 무엇을 하는가?
8. .login파일을 다음과 같이 편집하시오.

   ㄱ. 사용자를 맞이하시오.
   ㄴ. 자기의 홈등록부가 거기에 없다면 경로에 추가하시오.
   ㄷ. login파일을 원천화하시오.

9. path와 PATH의 차이점은 무엇인가?

련습문제 58: 리력

1. 등록탈퇴할 때 어느 파일에 리력사건이 기억되는가? 어느 변수가 표시되는 리력사건들의 개수를 조종하는가? savehist변수의 목적은 무엇인가?
2. 리력목록을 거꾸로 인쇄하시오.
3. 행번호없이 리력목록을 인쇄하시오.
4. 다음의 지령들을 입력하시오.

   ㄱ. ls-a
   ㄴ. date '+%T'
   ㄷ. cal 2000
   ㄹ. cat /etc/passwd

ㅁ. cd

5. history를 입력하시오. 출력이 어떻게 되는가?

   ㄱ. 마지막지령을 어떻게 재실행시키는가?
   ㄴ. echo a b c를 입력하시오.
      history지령을 리용하여 그의 마지막인수 c만을 가진 echo지령을 재실행시키시오.

6. 문자 d로 시작된 리력목록에 있는 마지막지령을 인쇄하고 실행시키기 위해 리력을 리용하시오.
7. c로 시작되는 마지막지령을 인쇄하시오.
8. 이전지령으로부터 echo지령과 마지막인수를 실행하시오.
9. 리력바꿔넣기지령을 리용하여 T를 date지령에서 H로 교체하시오.
10. 지령행편집을 위한 vi편집기를 기동하기 위해 bindkey지령을 어떻게 리용하는가?
11. 편집기지령들과 그 기능을 어떻게 표시하는가?
12. 편집건들이 실지로 어떻게 맺어 지는가 하는 것을 어떻게 아는가?
13. fignore변수의 기능을 설명하시오.

련습문제 59: 쉘메타문자

1. 재촉문에서 다음의것을 입력하시오.
   touch ab abc a1 a2 a3 all a12 ba ba.1 ba.2 filex filey Abc ABC Abc2 abc
2. 다음의것을 수행하는 지령을 작성하고 검사하시오.

   ㄱ. a로 시작하는 모든 파일을 표시하시오.
   ㄴ. 적어도 한개 수자로 끝나는 모든 파일들을 표시하시오.
   ㄷ. a나 A로 시작하지 않는 모든 파일들을 표시하시오.
   ㄹ. 뒤에 한개 수자가 오는 점으로 끝나는 모든 파일들을 표시하시오.
   ㅁ. 2개 문자만을 포함하는 모든 파일들을 표시하시오.
   ㅂ. 모든 문자들이 대문자인 3개 문자로 된 파일들을 표시하시오.
   ㅅ. 11 또는 12로 끝나는 파일들을 표시하시오.
   ㅇ. x 또는 y로 끝나는 파일들을 표시하시오.
   ㅈ. 한개 수자, 대문자 또는 소문자로 끝나는 모든 파일들을 표시하시오.
   ㅊ. ab를 포함하는 모든 파일들을 표시하시오.
   ㅋ. a로 시작하는 두개의 문자로 된 파일들을 지우기하시오.

련습문제 60: 방향바꾸기

1. 말단과 관련된 3개의 파일흐름들의 이름은 무엇인가?
2. 파일형식자란 무엇인가?
3. 다음의것을 하려면 어떤 지령을 리용하는가?

   ㄱ. ls지령의 출력을 lsfile파일에 방향바꾸기하려면?
   ㄴ. date지령의 출력을 lsfile파일에 방향바꾸기하고 추가하려면?
   ㄷ. who지령의 출력을 lsfile에 방향바꾸기하려면? 어떤 일이 생기는가?

4. cp만을 입력할 때 어떻게 되는가?

   ㄱ. 우의 실례에서 생기는 오유통보문을 파일에 어떻게 기억하는가?

5. find지령을 리용하여 어미등록부에서 시작하고 등록부형인 모든 파일들을 찾으시오. find라고 하는 파일에 표준출력을 기억하고 found.errs파일에 표준오유를 기억하시오.
6. noclobber는 무엇인가? 그것을 어떻게 무시하는가?
7. 3개의 지령의 출력을 gottemall파일에 방향바꾸기하시오.
8. ps와 wc지령과 함께 파이프를 리용하여 현재 몇개의 프로세스를 실행하고 있는가를 알아 내시오.

련습문제 61: 변수와 배렬

1. 국부변수와 환경변수의 차이는 무엇인가?
2. 모든 국부변수들을 어떻게 표시하는가? 환경변수들은?
3. 국부변수들을 어느 초기화파일에 기억시키는가? 왜?
4. fruit배렬을 작성하시오. 배렬에 5개 종류의 파일을 넣으시오.

   ㄱ. 배렬을 인쇄하시오.
   ㄴ. 배렬의 마지막요소를 인쇄하시오.
   ㄷ. 배렬에 있는 요소들의 개수를 인쇄하시오.
   ㄹ. 배렬에서 첫번째 요소를 지우기하시오.
   ㅁ. 파일이 아닌 항목을 배렬에 기억할수 있는가?

5. 단어목록과 문자렬의 차이를 설명하시오.

부록 1. 쉘프로그람작성자들을 위한 UNIX편의프로그람

at, batch-지령들을 후에 실행

at [-csm] [-f script] [-qqueue] time [date] [+increment]
at -l[job...]
at -r job...
batch

at와 batch는 후에 실행되여야 할 표준입력으로부터 지령들을 읽는다. at는 지령을 언제 실행하겠는가를 지정할수 있게 하는데 이때 batch로 대기된 일감들은 체계넣기준위가 허락할 때 실행한다. 후에 표준입력이나 파일로부터 지령을 읽어서 실행시킨다. 만일 방향바꾸기되지 않으면 출력을 사용자에게 보낸다.

실례 1

1. at 6:30 am Dec 12 < program
2. at noon tomorrow < program
3. at 1945 pm august 9 < program
4. at now +3 hours < program
5. at 8:30 Jan 4 < program
6. at -r 83883555320.a

설명

1. 12월 12일 아침 6시 30분에 일감을 시작한다.
2. 래일 낮에 일감을 시작한다.
3. 8월 9일 저녁 7시 45분에 일감을 시작한다.
4. 3시간안에 일감을 시작한다.
5. 1월 4일 아침 6시 30분에 일감을 시작한다.
6. 이미 계획된 일감 83883555320.a를 지우기한다.

awk-패턴읽기와 언어처리

awk [-fprogram-file] [-Fc] [prog] [parameters][filename]

awk는 prog에서 지정된 패턴의 모임과 일치하는 행들을 찾기 위한 매 입력파일을 조사한다.

실례 2

1. awk '{ print $1, $2 }' file
2. awk '/John/{ print $3 , $4 }' file
3. awk -F: '{ print $3 }' /etc/passwd
4. date | awk '{ print $6 }'

설명

1. 마당들이 공백으로 분리된 file의 첫 2개 마당을 인쇄한다.

2. 패턴 John이 나타나면 마당 3과 4를 인쇄한다.
3. 마당분리기호로서 두점을 리용하여 /etc/passwd의 세번째 마당을 인쇄한다.
4. date지령의 출력을 awk에 보내고 여섯번째 마당을 인쇄한다.

banner-포스타를 만들기

banner는 표준출력에 큰 문자로 인수(매개 인수는 10문자이상)들을 인쇄한다.

실례 3

```
banner   Happy   Birthday
```

설명

문자렬 Happy Birthday를 banner형식으로 표시한다.

basename-등록부이름으로 경로이름의 부분들을 넘기기.

basename string [suffix]
dirname string

basename은 빗선으로 끝나는 앞붙이와 뒤붙이를 문자렬로부터 지우고 그 결과를 표준출력에 인쇄한다.

실례 4

```
1  basename   / usr / local / bin
2  scripname = “ ‘ basename $0 □”
```

설명

1. 앞붙이 /usr/local를 없애고 bin을 현시한다.
2. 스크립트의 이름 $0를 변수 scriptname에 대입한다.

bc-정밀산수연산처리

bc [-c] [-l] [filename]

bc는 c와 류사한 언어에 대한 대화형처리기이지만 제한되지 않는 정밀산수연산을 제공한다. 그것은 주어 진 파일로부터 입력을 받은 다음 표준입력을 읽는다.

실례 5

```
1.  bc << EOF
    scale=3
    4.5+5.6/3
    EOF
    Output  :  6.366
-----------------------

2.  bc
    ibase=2
```

```
        5
        101  (Output)
        20
        10100  (Output)
    ^D
```

설명

1. 이것은 here문서이다. 첫 EOF로부터 마지막 EOF까지의 입력은 bc지령에 주어 진다. 척도는 소수점의 오른쪽에 놓이는 수의 개수를 지정한다. 계산결과는 현시장치에 표시된다.
2. 2진법이다. 수는 2진수로 변환된다.

bdiff-2개의 큰 파일을 비교

bdiff는 diff에 대해서 너무 큰 2개의 파일을 비교한다.

cal-력서를 현시

cal [[month] year]

cal은 지정된 년도에 대한 력서를 인쇄한다. 만약 달이 지정되면 달에 대한 달력도 인쇄된다. 어느것도 지정되지 않으면 현재달에 대한 달력만 인쇄한다.

실례 6

1. cal 1997
2. cal 5 1978

설명

1. 1997년도 력서를 인쇄한다.
2. 1978년 5월 달력을 인쇄한다.

cat-파일들을 련결시키고 현시.

cat [-bnsuvet] filename

cat는 순서대로 매 파일을 읽고 표준출력에 그것을 쓴다. 입력파일이 주어 지지 않거나 -인수가 나타나면 cat는 표준입력파일로부터 읽는다.

실례 7

1. cat / etc / passwd
2. cat -n file1 file2 >> file3

설명

1. /etc/passwd파일의 내용을 현시한다.
2. file1과 file2를 련결시켜 file3에 그 출력을 추가한다. -n추가선택은 매행에 번호를 붙이도록 한다.

chmod-파일허가방식변경

chmod [-fR] mode filename
chmod [ugoa] { + | - | = } [rwxlsStTugo] filename

chmod는 파일의 방식을 변화시키거나 대입한다. 파일의 방식은 그의 허가와 다른 속성들을 규정한다. 그 방식은 절대적이거나 상징적일수 있다.

실례 8

1. chmod +x script.file
2. chmod u+x , g-x file
3. chmod 755

설명

1. script.file에 있는 사용자, 그룹과 기타 다른것들에게 실행허가를 준다.
2. 사용자에게 실행허가를 주고 file에 있는 그룹으로부터 실행허가를 지우기한다.
3. 현재작업등록부에 있는 모든 파일들에서 사용자에 대해서는 읽기, 쓰기, 실행을 설정하고 그룹에 대해서는 읽기와 실행, 기타 다른것들에 대해서는 읽기와 실행을 설정한다. 그 값은 8진수(111 101 101)이다.

rwxxr-xr-x

chown-파일소유자변경

chown [-fhR] owner filename

chown은 파일소유자를 소유자로 변경시킨다. 소유자는 10진수리용자 ID나 /etc/passwd파일에 있는 등록가입이름(login name)일수 있다. 파일의 소유자(또는 주사용자)는 그 파일의 소유자를 변경시킬수 있다.

실례 9

1. chown john filex
2. chown -R ellie ellie

설명

1. filex의 사용자 ID를 john으로 변경시킨다.
2. 소유권(ownership)은 ellie등록부에 있는 모든 파일에 대하여 ellie로 재귀적으로 변경시킨다.

clear-말단현시장치 지우기

cmp-2개 파일의 비교.

cmp [l] [-s] filename1 filename2

2개의 파일을 비교한다. cmp는 파일들이 같으면 설명문을 만들지 않지만 차이나면 첫 차이점이 생기는 곳에서 바이트와 행번호를 알려 준다.

실례 10

cmp file.new file.old

설명

파일들이 차이나면 문자번호와 행번호를 현시한다.

compress- 압축, 풀기, zcat압축, 파일풀기 또는 확장된 파일들의 현시

compress [-cfv] [-b bits] [filename]
uncompress [-cv] [filename]
zcat [filename]

compress는 적응적인 Lempel-Ziv코드작성방법을 리용하여 이름 지어 진 파일들의 크기를 줄인다. 가능할 때마다 매 파일은 확장자가 .Z인 파일로 교체된다. 소유방식, 적분시간, 변경시간은 같아 진다. 만약 파일이 지정되지 않으면 표준입력은 표준출력으로 압축된다.

실례 11

1. compress -v book
   book : Compression : 35.07 % - - replaced with book.z
2. ls
   book .z

설명

1. book.z라는 파일로 book를 압축하고 그 파일을 압축한 퍼센트와 새로운 이름을 현시한다.

cp-파일의 복사

cp [-l] [-p] [-r] [filename] target

cp지령은 filename을 파일이나 등록부인 다른 대상으로 복사한다. 파일이름과 대상은 같은 이름을 가질수 없다. 만약 대상이 등록부가 아니면 그앞에 한개 파일만 지적될수 있다. 만일 등록부라면 하나이상의 파일들을 지정할수 있다. 대상이 존재하지 않으면 cp는 target라는 이름을 가진 파일을 만든다. 대상이 존재하고 등록부가 아니라면 그의 내용들에 겹쳐쓰기된다. 대상이 등록부이면 파일들을 그 등록부에 복사한다.

실례 12

1. cp file1 file2
2. cp chapter1 book
3. cp -r desktop / usr / bin / tester

설명

1. file1의 내용을 file2에 복사한다.
2. chapter1의 내용을 book등록부에 복사한다. book등록부에서 chapter1은 원

레이름을 가진다.
3. 전체 desktop등록부를 /usr/bin/tester에 재귀적으로 복사한다.

cpio-파일문서의 복사

cpio –i [bBcdfkmrsStuvV6] [-C bufsize] [-E filename]
[-H heade] [-l filename [-M message]] [-R id]
[pattern]
cpio –o [aABcLvV] [-C bufsize] [-H header]
[-O filename [-M message]]
cpio –p [adlLmuvV] [-R id] directory

보통 테프나 등록부에 여벌만들기(backup)를 하기 위하여 주어 진 변경자에 따라서 파일문서를 복사한다.

실례 13
find . -depth -print | cpio -pdmv / home / john / tmp
설명
find는 현재등록부에서 시작하여 등록부계층구조아래로 내려 가 그 등록부에 쓰기허가가 없을 때에도 등록부의 매 입력을 인쇄하고 /home부분에 있는 john/tmp등록부에 복사될 cpio으로 파일이름을 보낸다.

cron-박자데몬

cron은 지정된 날자와 시간에 지령들을 실행시킨다. 규칙적으로 일감일정작성을 /etc/crontab파일에 지정할수 있다. cron을 리용하자면 다음과 같은것들중의 하나가 참이여야 한다.

(1) 주사용자이다.
(2) 규칙적인 사용자이지만 사용자 ID는 /etc/cron.allow파일에 있다.
(3) 규칙적인 사용자이지만 체계는 비여 있는 파일 /etc/cron.deny를 가지고 있다.

crypt-파일의 암호화 또는 복호화

crypt [password]

crypt는 파일의 내용을 암호화하거나 복호화한다. 암호는 변환의 형태를 선택하는 열쇠이다.

cut-파일의 매 행에서 선택된 마당이나 문자의 지우기

cut –clist [filename]
cut –flist [-dc] [-s] [filename]

cut지령은 파일의 행으로부터 렬이나 문자를 없애 버린다. 파일이 주어 지지 않으면 표준입력을 사용한다. D추가선택은 마당구분기호를 지정한다. 기정으로 구분기호는 타브(tab)이다.

실례 14

1. cut -d: -f1,3 /etc/passwd
2. cut -d: -f1-5 /etc/passwd
3. cut -c1-3, 8-12 /etc/passwd
4. date | cut -cl?3

설명

1. 마당구분기호로서 두점을 사용하여 /etc/passwd파일의 마당 1과 마당 3을 현시한다.
2. 마당구분기호로서 두점을 리용하여 /etc/passwd의 마당 1부터 5까지를 현시한다.
3. /etc/passwd파일로부터 매행의 1부터 3까지와 8부터 12까지의 물음표들을 없애 버리고 현시한다.
4. 입력으로 date지령의 출력을 cut에 보낸다. 첫 3개 물음표를 인쇄한다.

date-날자와 시간의 현시 또는 날자의 설정

[-u] [-a [-] sss.fff] [yymmddhhmm [.ss]] [+format]

인수가 없이도 date지령은 날자와 시간을 현시한다. 만일 지령행의 인수가 +기호로 시작하면 그 인수의 나머지부분은 출력을 형식화(format)하는데 사용된다. 퍼센트기호를 사용하면 다음문자는 년이나 요일과 같은 날자의 특별한 부분을 뽑아 내기 위한 형식문자이다. 날자를 설정하기 위해서는 지령행인수를 년, 월, 일, 시간과 분을 나타내는 수자로 표현한다.

실례 15

1. date +%T
2. date +20$y
3. date "+It is now %m/%d/%y"

설명

1. 시간을 20:25:51로 현시한다.
2. 2096을 현시한다.
3. It is now 07/25/96을 현시한다.

diff- 두 파일차이의 비교

diff [-bitw] [-c | -C]

2개의 파일을 비교하여 행별로 차이를 현시한다. 또한 ed편집기를 리용하여 변화시키는 지령들을 현시한다.

실례 16

```
diff file1 file2
1c1    < hello thre
```

```
---
> Hello   there.
2a3
I ‘m   fine.
```

설명

file1과 file2의 매행이 어떻게 차이나는가 하는것을 보여 준다. 첫번째 파일은 <기호로 나타내고 두번째 파일은 >기호로 나타낸다. 매행앞에 파일들을 같게 하는데 리용되는 편집지령 ed가 놓인다.

du-디스크사용의 요약

du [-arskod] [name]

du지령은 지정된 매 등록부와 파일에 있는 모든 파일들과 (재귀적으로) 등록부에 있는 512byte블로크의 개수를 보여 준다.

실례 17

1. du -s / desktop
2. du -a

설명

1. /desktop와 그의 보조등록부에 있는 모든 파일들에 대한 블로크사용의 요약을 현시한다.
2. 등록부와 보조등록부에 있는 매 파일에 대한 블로크사용법을 현시한다.

echo-인수의 현시

echo [argument]..
echo [-n] [argument]

echo는 표준출력에 빈칸으로 분리되고 행바꾸기문자로 끝나는 인수들을 쓴다.

System V echo 추가선택:
\ b 뒤방향지우기
\ c 행바꾸기문자억제
\ f 페지바꾸기
\ n 행바꾸기문자
\ r 되돌이
\ t 타브
\ v 수직타브
\ \ 거꿀빗선
\ 0n n은 8진수값으로 1, 2, 3이다.

egrep-완전한 정규식을 리용한 패턴을 찾는 파일탐색

egrep [-bchilnsv] [-e special-expression] [-filename]

[strings] [filename...]

egrep(표현식 grep)는 문자들의 패턴(pattern)을 찾기 위한 파일들을 탐색하고 그 패턴을 포함하는 모든 행을 인쇄한다. egrep는 완전한 정규식(문자와 특수문자들의 완전모임(full set)을 리용하는 문자렬값에 의한 표현식)을 리용하여 그 패턴들을 대조한다.

실례 18

```
1. egrep  'Tom | John'  datafile
2. egrep  '^  [A-Z]+'  file
```

설명

1. 패턴 Tom이나 John을 포함하는 datafile의 모든 행들을 현시한다.
2. 하나이상의 대문자로 시작하는 모든 행들을 현시한다.

expr-표현식으로 인수들을 평가

expr argments

인수들을 표현식으로서 취급한다. 평가후 그 결과를 표준출력에로 쓰기한다. 표현식의 항들은 공백으로 분리하여야 한다. 쉘에서 특수한 문자들을 탈퇴시켜야 한다. 간단한 산수연산을 진행하기 위하여 Bourne쉘스크립트에서 사용된다.

실례 19

```
1. expr  5 + 4
2. expr  5 \ * 3
3. num = 0
   num = ' expr  $num + 1 '
```

설명

1. 5+4의 합을 인쇄한다.
2. 5*3의 결과를 인쇄한다. 별표는 쉘확장으로부터 보호된다.
3. 0을 변수 num에 대입한 다음 expr지령은 1을 num에 더하고 결과를 num에 대입한다.

fgrep-문자렬을 위한 파일탐색

fgrep [-bchilnsvx] [-e string] [-f filename] [strings] [filename...]

fgrep(fast grep)는 문자렬을 위하여 파일들을 탐색하고 그 문자렬을 포함하는 모든 행들을 인쇄한다. fgrep는 문자로서 정규표현메타문자를 해석하기때문에grep와 egrep와는 다르다.

실례 20

```
1. fgep  '***'  *
2. fgrep  '[  ]  *  ?  $'  filex
```

설명

1. 현재등록부에 있는 매 파일로부터 3개의 별표를 포함하는 임의의 행을 현시한다. 모든 문자들은 그자체로서 취급된다. 즉 메타문자들은 특별한 의미를 가지지 않는다.
2. 인용부호안에 있는 문자렬을 가진 filex에 있는 행들을 현시한다.

file-파일의 내용을 조사하여 파일형태의 결정

file [[-f ffile] [-cl] [-m mfile] filename...

file은 파일이 무엇을 포함하는가를 결정하기 위하여 매 파일이름에 대한 련속적인 검사를 진행한다. 만일 파일의 내용이 ASCII본문으로 나타나면 file의 첫 512byte를 검사하여 그 언어를 추측하려고 한다.

실례 21

1. file bin / ls
   / bin/ ls : sparc pure dynamically linked executable
2. file go
   go: executable shell script
3. file junk
   junk : English text

설명

1. ls는 실행될 때 동적으로 련결된 2진파일이다.
2. go는 실행가능한 쉘스크립트이다.
3. junk는 ASCII본문을 포함하는 파일이다.

find-파일찾기

find path -name-list expression

find는 추가선택들을 대조하는 파일들을 찾는 경로이름목록(즉 하나이상의 경로이름)에서 매 경로이름에 대하여 등록부계층구조로 재귀적으로 내려 간다. 첫 인수는 탐색을 시작하는 경로이다. 나머지 인수들은 이름, 크기, 소유자, 허가 등과 같은 파일들을 찾기 위한 몇가지 기준을 지정한다. 서로 다른 문법에 대해서는 UNIX사용지도서페지(manual pages)를 검사해야 한다.

실례 22

1. find . -name \ * . c - print
2. find . .-type f - print
3. find . -type d -print
4. find / -size o -exec rm “{}” \ ;
5. find ~ -perm 644 -print
6. find . -type f -size +500c -atime +21 -ok rm -f : {} 刼 \ ;
7. find . -name core -print 2> / dev/ null (Bourne and Kon Shells)

(find~ . -name core -print > / dev / tty) >& / dev / null (C shell)
8. find / - user ellie xdev -print
9. find ~ -atime +31 -exec mv {} / old / {} \ ; -print

설명

1. 현재작업등록부(.)에서 시작하여 .c로 끝나는 모든 파일을 찾고 파일의 완전한 경로이름을 인쇄한다.
2. 어미등록부(. .)에서 시작하여 file형식의 모든 파일들 즉 등록부가 아닌 파일들을 찾는다.
3. 현재등록부(.)에서 시작하여 모든 등록부파일들을 찾는다.
4. 뿌리등록부에서 시작하여 크기가 령인 모든 파일들을 찾아서 그것들을 지우기한다. { }는 매 파일이름들의 공간채우기(place holder)로 사용한다.
5. 사용자의 홈등록부 ~(Korn쉘과 C쉘)에서 시작하여 허가 644(소유자에 대해서는 읽기와 쓰기허가, 그룹과 기타에 대해서는 읽기허가)를 가지는 모든 파일을 찾는다.
6. 현재작업등록부에서 시작하여 500byte를 넘으며 마지막 21일에 호출되지 않은 파일을 찾고 그 파일을 지워도 되는가를 물어 본다.
7. 현재작업등록부에서 시작하여 core라는 모든 파일들을 찾아서 현시하고 오유들을 /dev/null 즉 UNIX비트바키트로 보낸다.
8. 사용자 ellie에 속하는 root구획(partition)에 있는 모든 파일을 인쇄한다.
9. 31일보다 더 오랜 파일들을 등록부 /old에 이동시키고 그것들을 이동시킬 때 파일들을 인쇄한다.

finger-국부및 원격사용자에 대한 정보현시

finger [-bfhilmpqsw] [username...]

finger [-l] username @ hostname...

암시적으로 finger지령은 매 등록가입된(logged-in) 사용자에 대한 정보들을 현시한다. 이 정보는 즉 등록가입(login)이름, 옹근이름(full name), 말단이름(쓰기허가가 거절되면 그앞에 *가 붙는다.), 휴식(idle)시간, 등록가입(login)시간 그리고 그 위치가 알려 지면 그것을 포함한다.

fmt-간단한 본문양식기(formatter)

fmt [-c] [-s] [-w width] [inputfile]

fmt는 -w너비추가선택에서 지정된 문자들에 있는 수자와 같은 개수의 출력행을 만들기 위하여 행들을 채우고 련결하는 간단한 본문양식기(formatter)이다. 기정값으로서 너비는 72이다. fmt는 인수로서 목록화된 입력파일들을 련결한다. 만일 아무것도 주어지지 않으면 fmt는 표준입력으로부터 본문을 형식화(format)한다.

실례 23

fmt -c -w45 letter

설명

letter를 형식화(format)한다. -c추가선택은 단락안에 있는 첫 2개 행의 들여쓰기를 보존하고 그다음의 매행들의 왼쪽 경계를 두번째 행의 경계에 맞춘다. -w추가선택은 출력행의 렬수를 45개로 한다.

Fold-긴 행의 접기

fold [-w width] [-width] [filename]

지정된 파일의 내용을 접거나 만일 파일이 지정되지 않으면 표준입력은 행들을 나누어 너비가 최대로 되게 한다. 기정적인 너비값은 80이다. 만일 타브(tab)가 존재한다면 너비는 8의 배수로 되여야 하거나 타브(tab)가 확장되여야 한다.

ftp-파일전송프로그람

ftp [-dgintv] [hostname]

ftp지령은 인터네트표준파일전송규약(FTP)에 대한 사용자대면부이다. ftp는 파일들을 원격망싸이트(remote network site)에로 혹은 그곳으로부터 전송한다. 파일전송프로그람은 UNIX콤퓨터에만 국한되지 않는다.

실례 24

1. ftp ftp.uu.net
2. ftp -n 127.150.28.56

설명

1. 큰 자료저장고(repository)인 콤퓨터 ftp.uu.net에 대한 ftp는 UNIX체계를 위해 전자우편과 새소식을 처리하는 UUNET봉사(service)에 의해 실행된다.
2. 127.45.4.1에서 그 콤퓨터에 접속하고 자동등록가입(auto-login)은 하지 않는다.

getopt(s)-지령행추가선택구문해석

getopts지령은 getopt지령을 대신한다. getopts를 쉘수속(shell procedure)에 의한 구문해석을 쉽게 하기 위하여 지령행에서 추가선택들을 갈라 내여 합법적인 추가선택들인가를 검사하는데 사용한다.

grep-패턴에 대한 파일탐색

grep [-bchilnsvw] limited -regular -expression [filename ...]

grep는 패턴을 찾기 위하여 파일을 탐색하고 그 패턴이 있는 모든 행들을 인쇄한다. 정규표현메타문자들을 사용하여 패턴들을 대조한다. egrep는 확장된 메타문자모임을 가진다.

실례 25

1. grep Tom file1 file2 file3

```
2. grep -in '^tom savage *
```

설명

1. grep는 패턴 Tom이 있는 file1, file2, file3의 모든 행들을 현시한다.
2. grep는 만일 tom savage가 행의 시작에 앉으면 현재작업등록부의 tom savage가 있는 파일의 모든 행들을 행번호와 함께 현시하고 선택은 무시한다.

groups-사용자그룹성원 인쇄

groups [user...]

지령 groups는 표준출력에 사용자 혹은 임의로 지정된 사용자가 속하여 있는 그룹을 인쇄한다.

id-사용자이름, 사용자 ID, 그룹이름과 그룹 ID의 인쇄

/usr/bin/id [-a]

id는 사용자 ID, 사용자이름, 그룹 ID, 그룹이름을 현시한다. 만일 실지ID와 유효ID가 대조하지 않으면 둘다 인쇄한다.

jsh-표준,일감조종쉘(shell)

jsh [-acefhiknprstuvx] [argument]

지령jsh는 Bourne 쉘의 모든 기능을 제공하며 일감조종을 할수 있게 하는 표준 Bourne쉘에 대한 대면부이다.

line-한행 읽기

line은 표준입력으로부터 한개 행(행바꾸기문자까지)을 읽고 그것을 표준출력에 쓴다. EOF에서 행의 탈퇴코드(exit code)를 1로 되돌리며 적어도 한개의 행바꾸기를 항상 인쇄한다. 보통 쉘파일에서 사용되여 사용자의 말단으로부터 읽는다.

logname-처리를 진행하는 사용자이름 얻기

lp 출력을 인쇄기로 보내기(AT&T)

lp [-cmsw] [-d dest] [-number] [-o option] [-t title] filename...
cancel [ids] [printers]

lp, cancel은 요구를 행인쇄기에 보내거나 취소한다.

실례 26

```
1. lp -n5 filea fileb
2. lp -dShakespeare filex
```

설명

1. filea와 fileb의 5개의 사본을 인쇄기에 보낸다.
2. filex를 인쇄할 인쇄기로서 Shakespeare를 지정한다.

lpr-출력을 인쇄기로 보내기(UCB)

lpr [-Pprinter] [-#copies] [-Cclass] [-Jjob]
[-Ttitle] [-i [indent]] [-1234font] [-wcols]
[-r] [-m] [-h] [-s] [-filter-option]
[filename...]

lpr는 수단이 리용가능하면 다음인쇄를 위한 입출구완충구역에서 인쇄일감을 작성한다. 매 인쇄기일감은 조종일감과 하나이상의 자료파일들로 되여 있다.

실례 27

1. lpr -#5 filea fileb
2. lpr -PShakespeare filex

설명

1. filea과 fileb의 5개의 사본을 인쇄기에 보낸다.
2. filex를 인쇄할 인쇄기로서 Shakespeare를 지정한다.

lpstat-LP인쇄기봉사의 상태정보 인쇄(AT&T)

lpq- 인쇄기상태정보 인쇄(UCB)

ls-등록부의 내용 표시

ls [-abcCdfFgilLmnopqrRstux1] [names]

매 등록부인수에 대해서 ls는 등록부의 내용을 표시한다. 매 파일인수에 대해서 ls는 파일이름과 요구되는 다른 정보를 반복한다. 출력은 암시적으로 자모순서로 정렬된다. 인수가 주어 지지 않으면 현재등록부를 현시한다.

실례 28

1. ls -alF
2. ls -d a*
3. ls -i

설명

1. -a는 보이지 않는 파일(점으로 시작하는 파일)들을 표시하고 -l은 파일의 속성들을 보여 주는 긴 표시이며 -F는 등록부파일이름의 끝에 빗선을 붙이며 *는 실행가능한 스크립트이름의 끝에, @기호는 기호적으로 련결된 파일의 끝에 놓인다.
2. 만일 -d추가선택에 대한 인수가 등록부라면 등록부의 이름만을 표시하고 내용은 표시하지 않는다.
3. -i추가선택은 매 파일이름앞에 그것의 inode번호가 놓이게 한다.

mail-mail, rmail-우편읽기 또는 사용자에게 보내기

Sending mail
  mail [-tw] [-m message_type] recipient...
  rmail [-tw] [-m message_type] recipient...
Reading mail
  mail [-ehpPqr] [-f filename]
Forwarding mail
  mail -F recipient...
Debugging
  mail [-x debug_level] [other_mail_options] recipient...
  mail [-T mailsurr_file] recipient...

수신자는 보통 login에 의해 인식되는 사용자이름이다. 수신자의 이름을 지울 때 우편은 통보문이 보내지기를 요구한다. 그것은 표준입력으로부터 파일의 끝(Ctrl-D)까지 읽거나 만일 말단으로부터 읽으면 점으로 이루어 진 행을 읽을 때까지 읽는다. 그 지적자들중 어느 하나가 접수되면 mail은 그 문자를 매 수신자에 대한 우편파일에 추가한다.

mailx-체계를 처리하는 대화형통보문처리체계

mailx [-deHilnNUvV] [-f [filename | + folder]]
[-T filename] [-u user] [recipient...]
mailx [-DddfFfI]

우에서 표시한 우편편의프로그람들은 우편통보문을 받고 보내고 조작하기 위한 대화형대면부를 제공한다. 기본망편의프로그람들은 몇개의 수단들을 리용할수 있도록 설치되여야 한다. 들어 오는 우편은 mailbox파일에 기억되고 그것이 읽어 진후 mbox파일에 보내진다.

make-관련된 프로그람들과 파일들의 그룹을 유지, 갱신 그리고 재생

make [-f makefile] [-d] [-dd] [-D]
[-DD] [-e] [-i] [-k] [-n] [-p]
[-P] [-q] [-r] [-s] [-S] [-t] [target]
[macro=value]

make는 설명파일에 표시된 지령들에 따라서 파일들을 갱신하고 목적파일이 같은 이름의 기존파일보다 더 새로운것이면 목적파일을 갱신한다.

mesg-쓰기지령으로부터 생기는 통보문의 허가 또는 거절

mseg [-n] [-y]

인수 –n을 가진 mesg는 사용자말단에 대한 비사용자쓰기허가를 취소하여 write에 의한 통보문을 금지시킨다. 인수 –y가 있는 mesg는 허가를 회복한다. mesg는 그것을 변화시키지 않고 현재상태를 알려 준다.

mkdir-등록부를 창조한다.

mkdir [-p] dirname

more-본문파일을 통한 열람 또는 페지

more [-cdflrsuw] [-lines] [+linenumber] [+/pattern]
[filename]
page [-cdflrsuw] [-lines] [+linenumber] [+/pattern]
[filename]

more는 말단에서 본문파일의 내용을 한번에 한개 화면씩 현시하는 려과기이다. 그것은 보통 매 화면마다 정지하여 화면의 밑에 -More-를 인쇄한다.

mv-파일의 이동 또는 변경

mv [-f] [-i] filename1 [filename2] target

mv지령은 원천파일이름을 목적파일이름으로 이동시킨다. 파일이름과 목적파일의 이름은 같지 않을수 있다. target가 등록부가 아니면 그앞에 한개의 파일만이 지정될수 있다. 등록부에는 한개이상의 파일이 지정될수 있다. target가 존재하지 않으면 mv는 target라고 하는 파일을 작성한다. target가 존재하고 등록부가 아니면 그 내용에 덧씌워 진다. target가 등록부이면 파일들이 그 등록부로 이동된다.

실례 29

1. mv file1 newname
2. mv -i test1 test2 train

설명

1. file1을 newname으로 이름을 고친다. newname이 존재하면 그 내용이 겹쳐쓰기된다.
2. 파일 test1과 test2를 train등록부에로 이동시킨다. -i추가선택은 대화형방식을 위한것인데 파일을 이동하기전에 물어 본다.

nawk-패턴조사와 언어처리

nawk [-F re] [-v var=value] [prog□] [filename]
nawk [-F re] [-v var=value] [-f progfile] [filename]

nawk는 패턴들의 모임과 대조하는 행들을 찾기 위하여 매개의 입력파일이름을 조사한다. 지령문자렬은 단일인용부호안에 놓여 쉘로부터 보호되여야 한다. awk프로그람은 파일, 파이프 또는 표준입력으로부터 지적된 정보를 려과하는데 리용되는 pattern/action명령문의 모임으로 이루어 진다.

newgrp-새로운 모임에 등록가입

newgrp [-] [-group]

newgrp는 사용자를 그의 실지ID와 효률적그룹 ID를 분화시켜 새로운 그룹에 등록한다. 사용자는 등록가입되여 있고 현재등록부는 변하지 않는다. newgrp의 실행은 지령이 오유로 끝날 때에도 현재쉘을 새로운 쉘로 교체한다.

news-새로운 항목들의 인쇄

news [-a] [-n] [-s] [items]

news는 현재사용자에게 사건들을 알려 주는데 리용된다. 습관상 이 사건들은 등록부 /var/news에 있는 파일에 의해 서술된다. 인수가 없이 호출될 때 news는 /var/news에 있는 모든 현재파일들의 내용을 인쇄하는데 제일 최근의것은 첫번째로 그리고 앞에 적당한 머리부를 붙여서 인쇄한다.

nice-낮은 우선권에서 지령의 실행

nice [-increment] commmand [agments]

/usr/bin/nice는 우선권의 순서를 짜는 속도가 보다 낮은 CPU일정작성우선권을 가지고 지령을 실행한다. 호출프로쎄스(일반적으로 사용자의 쉘)는 시분할일정작성등급에 있어야 한다. 그 지령은 시분할등급에서 실행된다. 기정증가값은 10이다. 증가값은 주사용자가 아니면 1부터 19까지의 범위에 있어야 한다. csh내부지령을 볼것.

nohup-지령중지와 정지무시

/usr/bin/nohup command [argments]

nohup에 대한 3개의 구별되는 판본들이 있다. nohup는 C쉘에 내장되여 있고 Bourne쉘을 리용할 때 /usr/bin/nohup에서 리용할수 있는 실행프로그람이다. 판본의 nohup는 지령들을 실행시켜 그것들이 HUP(중지)신호와 TERM(완료)신호들을 무시하게 한다. 표준출력이 말단이면 그것은 파일 nohup.out에 방향바꾸기된다. 우선권은 5만큼 증가된다. nohup는 쉘이 다음사용자로부터의 새치기나 입력에 응답하지 않도록 그로부터 &와 함께 호출되여야 한다.

실례 30

nohup lookup &

설명

lookup프로그람은 배경판(background)에서 실행되고 사용자가 등록탈퇴한다고 해도 완성될 때까지 계속 실행된다. 발생된 출력이 nohup.out라는 등록부에 있는 파일에 간다.

od-8진수표시

od [-bcCDdFfOoSsvXx] [filename] [[+] offset [.] [b]]

od는 첫번째 인수로 선택될 때 한개이상의 형식으로 파일이름을 현시한다. 만일 첫 인수가 빠지면 –o가 기정으로 된다. 실례로 파일은 8진수, ASCII, 10진수, 16진수로 현시될수 있다.

pack-pack, pcat, unpack-파일압축과 확장

pack [-] [-f] name...
pcat name...
unpack name...

pack는 파일을 압축한다. 매 입력파일이름은 필요할 때마다 name과 같은 호출방식,

호출날자, 변경된 날자와 소유자로 묶음파일 name.z에 의해 교체된다. 대표적으로 본문파일은 보통 크기의 60~75%로 줄어 든다. pcat는 그것이 려과기로서 리용될수 없다는것을 제외하고는 cat가 보통 파일에 대하여 할수 있는것을 묶음파일들에 대하여 한다. 지정된 파일이 풀어 져 표준출력에 씌여 진다. 결과 name.z라는 이름을 가진 묶음파일을 보려면 pcat name.z나 pcatname을 리용한다. unpack는 pack에 의해 작성된 파일을 확장한다.

passwd-등록가입암호와 암호속성변경

passwd [name]
passwd [-d] [-f] [-n min] [-w wan] [-x max] name
passwd -s [-a]
passwd -s [name]

passwd지령은 암호를 변경시키거나 사용자의 등록가입이름과 관련된 암호속성들을 표시한다. 추가적으로 특권을 가진 사용자들은 passwd를 리용하여 암호들과 등록가입이름과 관련된 속성을 설치하거나 변화시킬수 있다.

paste-여러개의 파일들이나 한개 파일의 련속적인 행들의 합치기

paste filename1 filename2...
paste -d list filename1 filename2...
paste -s [-d list] filename1 filename2

paste는 주어 진 파일들인 filename1과 filename2의 대조하는 행들을 련결한다. 그것은 표의 렬 또는 렬들로서 매 파일들을 취급하며 수평으로 그것을 붙인다.

실례 31

1. ls | paste - - -
2. paste -s -d “ \ t \ n “ testfile1 testfile2
3. paste file1 file2

설명

1. 파일들은 3개의 렬로 표시되고 한개 타브에 의해 련결된다.
2. 구분기호로서 타브나 행바꾸기을 리용하여 한쌍의 행들을 단일행으로 조합시킨다. 례를 들어 첫번째 쌍의 행들은 타브에 의해 련결되며 다음쌍들은 행바꾸기에 의해 련결된다. 그다음 쌍들은 한개 타브에 의해 련결되는 식으로 계속된다. -s추가선택은 testfile1에로부터 다음의 행들이 먼저 붙어 지고 그다음 testfile2로부터 행들이 붙도록 한다.
3. file1로부터의 한 행이 file2의 한행에 붙고 파일의 행들이 2개의 렬로서 나타나도록 타브에 의해 련결된다.

pcat-(pack를 볼것)

pg-파일을 한번에 한페지씩 현시

pg [-number] [-p string] [-cefnrs] [+linenumber]
[+/pattern/] [filename...]

pg지령은 말단에 한번에 한화면씩 파일이름들을 현시하게 하는 려과기이다. 만일 어떤 파일이름도 지적되지 않거나 파일이름 –가 나타나면 pg는 표준입력으로부터 읽는다. 매 화면이 나타난 다음에 재촉문이 나타난다. 만일 사용자가 Return을 입력하면 다른 페지가 표시된다. 그것은 이미 지나간것을 복사하고 다시 보게 한다.

pr-파일을 인쇄

pr [[-columns] [-wwidth] [-a] [-eck] [-ick] [-drtfp]
[+page] [-nck] [-ooffset] [-llength] [-sseparator]
[-hhead] [-F] [-filename...]
pr [[- m] [-wwidth]] [-eck] [-ick] [-rtfp] [+page] [-nck]
[-ooffset] [-llength] [-sseperator] [-hheader] [-F]
[filename1 filename2...]

pr지령은 서로 다른 초기화항목에 따라 파일의 내용을 초기화하고 인쇄한다. 목록화된 출력은 stout에 보내여지고 파일이 마지막으로 수정된 페지번호, 날자 그리고 시간이 앞에 붙은 페지들로 분리된다. 추가선택이 지적되지 않으면 기정파일형식은 5개 행의 머리부와 5개 행의 꼬리부를 가진 66개 행으로 된다.

실례 32

```
pr  -2dh  “TITLE“  file1  file2
```

설명

file1과 file2에 대한 머리부 “TITLE”이 붙은 2개의 렬을 인쇄한다.

ps-프로쎄스상태의 통보

ps [-acdefil] [-g grplist] [-p proclist]
[-s sid list] [-t term] [-u uidlist]

ps는 능동프로쎄스에 대한 정보를 인쇄한다.

추가선택이 없으면 ps는 조종말단과 관련된 프로쎄스에 대한 정보들을 인쇄한다. 출력은 프로쎄스 ID, 말단식별자, 축적된 실행시간, 지령이름을 포함한다. 그렇지 않으면 현시된 정보는 그 추가선택들에 의하여 조종된다. Ps추가선택들은 AT&T와 UNIX의 Berkely형판본과 같지 않다.

실례 33

```
1.  ps  -aux  |  grep  ‘^linda‘   #  ucb
2.  ps  -ef  |  grep  ‘^linda‘   #  at&t
```

설명

1. 진행하고 있는 모든 프로쎄스들을 인쇄하고 출력을 grep프로그람에 넘기며 사용자 linda가 소유한 프로쎄스들만을 인쇄하는데 사용자 linda는 매행의 시작점에 있다(UCB판본에서만 가능).
2. 첫 실례와 같은데 다만 AT&T판본에서 진행된다.

pwd-현재작업등록부이름의 현시

rcp-원격파일복사

rcp [-p] filename1 filename2
rcp [-pr] filename1...directory

rcp지령은 다음의 형식으로 콤퓨터들사이에 파일들을 복사한다.

remothostname:path
user@hostname:file
user@hostname.domainname:file

실례 34

1. rcp dolphin : filename / tmp / newfilename
2. rcp filename broncos : newfilename

설명

1. 떨어 져 있는 콤퓨터 dolphin으로부터 이 콤퓨터에 있는 /tmp/newfilename에 filename을 복사한다.
2. 이 콤퓨터로부터 떨어 져 있는 콤퓨터 broncos에 filename을 복사하고 그것을 newfilename으로 이름 짓는다.

rlogin-원격등록가입

rlogin [-L] [-8] [-ec] [-l username] hostname

rlogin은 말단으로부터 hostname이라는 원격콤퓨터에게 원격등록가입대화조종(ssesion)을 설치한다. 호스트이름은 host의 자료기지안에 목록화되여 있는데 그 자료기지는 /etc/hostfile파일, Network Information Service(NIS) host배치표, 인터네트구역이름봉사기 또는 이것들의 조합에 포함될수 있다. 매 호스트는 하나의 사무이름(자료기지의 입구점안에 있는 첫번째 이름)과 선택적으로 하나이상의 별명을 가진다. 사무호스트이름이나 별명이 hostname으로 지정될수 있다. hostname의 목록은 콤퓨터의 파일 /etc/host.equiv에 기억될수 있다.

rm-등록부로부터 파일의 지우기

rm [-f] [-i] filename
rm -r [-f] [-i] dirname...[filename...]

rm은 파일이 쓰기허가되여 있으면 등록부로부터 한개이상의 파일들에 대한 입구점들을 지우기한다. 만일 파일이름이 기호적련결인 경우에 련결은 지우기되지만 그것이 표시하는 파일이나 등록부는 지워 지지 않는다. 등록부에서 쓰기허가를 가진다면 사용자는 그것을 지우기 위해 기호련결에 대한 쓰기허가를 가질 필요가 없다.

실례 35

1. rm file1 file2
2. rm -i
3. rm -rf dir

설명

1. file1과 file2를 등록부에서 지우기한다.
2. 현재작업등록부에 있는 모든 파일들을 지우기하지만 먼저 그렇게 해도 되는가를 문의한다.
3. dir아래에 있는 파일과 등록부를 재귀적으로 지우기하고 오유통보문을 무시한다.

rmdir-등록부의 지우기

rmdir [-p] [-s] dirname

등록부가 비여 있다면 그것을 지우기한다. *-p*를 리용하면 어미등록부들도 지우기된다.

rsh-원격쉘의 시동

rsh [-n] [-l username] hostname command
rsh hostname [-n] [-l username] command

rsh는 지정된 호스트이름에 련결하고 지정된 지령을 실행한다. rsh는 자기의 표준입력을 원격지령에, 원격지령의 표준출력을 자기의 표준출력에 그리고 원격지령의 표준오유를 자기의 표준오유에 복사한다. 새치기, 탈퇴, 완료신호는 원격지령에 전달되며 rsh는 보통 원격지령이 끝날 때 끝난다. 만일 지령이 주어 지지 않으면 rsh는 사용자를 rlogin을 리용하여 원격호스트에 가입시킨다.

실례 36

1. rsh bluebird ps -ef
2. rsh -l john owl ls ; echo $PATH ; cat .profile

설명

1. 콤퓨터 bluebird에 련결하고 그 콤퓨터에서 진행하는 모든 프로쎄스들을 현시한다.
2. 사용자 john으로 원격콤퓨터 owl에 가서 이 3가지 지령들을 모두 실행한다.

ruptime-국부콤퓨터의 호스트상태의 현시

ruptime [-alrtu]

ruptime은 국부망에서 매 콤퓨터에서의 작용시간(uptime)과 같은 상태를 주는데 이것들은 1분에 한번 망(network)에서 매 호스트에 의하여 파케트전송으로부터(packets broadcast) 형성된다. 상태보고를 5분동안 받지 않는 콤퓨터들은 떨어 진것으로 본다. 보통 목록화는 hostname에 의해 분류되지만 이 순서는 ruptime의 추가선택들중의 어느 하나를 지정하여 변화시킬수 있다.

rwho-국부콤퓨터에 등록가입된 사용자

rwho [-a]

rwho지령은 who와 류사한 출력을 만들어 내지만 그것은 망우에 있는 모든 콤퓨터들에 대하여 출력을 만든다. 그러나 그것은 관문을 통해서 하지 않고 호스트는 실행되는 rwho데몬뿐만아니라 등록부 /var/spool/rwho를 가져야 한다. 만일 콤퓨터로부터 5분동안에 통보를 받지 못하면 rwho는 콤퓨터가 떨어 졌다고 판단하고 그 콤퓨터에 마지막으로 가입된 사용자를 알려 주지 않는다. 만일 사용자가 1분이상 체계에 입력하지 않았다면 rwho는 이 휴식시간을 통보한다. 만일 사용자가 1시간이상 체계에 입력하지 않았다면 -a기발이 주어 지지 않을 때 사용자는 rwho출력으로부터 빠진다.

script-말단대화조종(terminal session)의 typescript의 창조

script [-a] [filename]

script는 모든 typescript가 말단에서 인쇄되게 한다. typescript는 파일이름에 씌여진다. 만일 파일이름이 주어 지지 않으면 typescript는 typescript라는 파일에 기억된다. 쉘이 탈퇴하거나 control-D가 입력될 때 스크립트가 끝난다.

실례 37

1. script
2. script myfile

설명

1. 새로운 쉘에 있는 스크립트대화조종을 시동한다. 말단에 현시된 모든것은 typescript라는 파일에 기억된다. 대화조종을 끝내려면 ˆd을 누르거나 탈퇴해야 한다.
2. 새로운 쉘에 있는 스크립트대화조종을 시동하여 말단에 표시된 모든것을 myfile에 기억한다. 대화조종을 끝내려면 ˆd를 누르거나 탈퇴하여야 한다.

sed-흐름선편집기

sed [-n] [-e script] [-f sfilename] [filename...]

sed는 이름 지어 진 파일이름을 표준출력에 복사하는데 지령의 스크립트에 따라 편집된다. 초기파일은 변화시키지 않는다.

실례 38

1. sed ‘ s / Elizabeth / Lizzy / g ‘ file
2. sed ‘ Dork / d ‘ file
3. sed -n ‘ 15 , 20p ‘ file

설명

1. 파일에서 모든 Elizabeth변수를 Lizzy로 교체하고 말단현시장치에 표시한다.
2. Dork를 포함하고 있는 모든 행들을 지우기하고 남아 있는 행들을 현시장치에 표시한다.
3. 15부터 20까지의 행들만 인쇄한다.

size-목적파일의 바이트분구크기를 인쇄

size [-f] [-f] [-n] [-o] [-V] [-x] filename...

size지령은 토막이나 부분크기의 정보를 ELF나 COFF목적파일에 적재된 매 분구에서 바이트로 생성한다. size는 본문, 자료와 bss(초기화되지 않은 자료)토막과 그 전체의 크기를 인쇄한다.

sleep-몇초동안 실행을 중지

sleep time

sleep는 몇초동안 실행을 중지한다. 이 지령은 일정한 시간후에 지령을 실행시키는데 리용된다.

실례 39

```
1. (sleep 105; command)
2. (In Script)
     while true
     do
       command
       sleep 60
         done
```

설명

1. 10s후에 지령이 실행된다. 재촉문은 즉시 귀환된다.
2. 순환을 시작한다. 지령을 실행하고 순환을 다시 시작하기전에 1분동안 중지한다.

sort-파일들의 정렬과 합치기

sort [-cmu] [-ooutput] [-T directory] [-ykmem]
[- dfiMnr] [-btx] [ ]pos1 [-pos2]] [filename...]

sort지령은 이름을 가진 모든 파일들의 행들을 정렬시키고 그 결과를 표준출력에 쓴다. 비교는 입력의 매행으로부터 추출된 하나 혹은 그이상의 정렬건에 기초하고 있다. 기정적으로 한개의 정렬건과 전체 입력행이 있는데 정렬은 순서대로 집합하는 기계에서 바이트단위의 사전식편집이다.

실례 40

```
1. sort filename
2. sort -u filename
3. sort -r filename
4. sort +1 -2 filename
5. sort -2n filename
6. sort -t: +2n -3 filename
7. sort -f filename
```

8. sort -b +1 filename

설명

1. 자모순서로 행들을 정렬한다.
2. 중복된 입력항목들을 정렬한다.
3. 거꾸로 정렬한다.
4. 행의 마지막까지 정렬하지 않고 마당 1에서 시작하고 마당 2에서 정지하여 정렬한다.
5. 세번째 마당을 수자적으로 정렬한다.
6. 마당 2부터 마당 3까지 마당분리기(-t:)로 표시된 옹근두점을 리용하여 수자들을 정렬한다.
7. 대문자와 소문자에 관계없이 정렬한다.
8. 마당 1에서부터 분류하여 처음에 오는 빈칸들을 지우기한다.

spell-맞춤법오유찾기

spell [-blvx] [-d hlist] [-s hstop] [+local_file] [filename]...

spell은 파일이름에서 단어를 모아 맞춤법목록에서 그것들을 찾는다. 맞춤법목록의 단어가운데서 나타나지 않거나 그 단어로부터 끌어 낼수 없는 단어는 표준출력에 인쇄한다. 만일 파일이름이 이름 지어 지지 않으면 단어들은 표준입력으로부터 모아 진다.

split-파일들을 부분으로 나누기

split [-n] [filename [name]]

split는 filename을 읽어서 n개 행으로 출력파일의 모임에 써넣는다. 첫 출력파일의 이름에는 aa가 붙고 이런식으로 zz까지 편집된다(최대파일수는 676개이다.). name의 최대길이는 파일체계에서 허락된 최대파일이름보다 2문자 작다. statvf를 볼것. 출력이름이 주어 지지 않으면x를 암시적으로 사용한다(출력파일은 xaa, xab 등으로 부른다.).

실례 41

1. split -500 filea
2. split -100 fileb out

설명

1. filea를500개의 행파일들로 나눈다. 파일들은 xaa, xab, xac등으로 이름 지어 진다.
2. fileb를1000개의 행파일들로 나누고 이름을 out.aa, out.ab등으로 짓는다.

strings-한개의 목적파일 또는 2진파일에서 인쇄할수 있는 문자렬탐색

strings [-a] [-o] [-number] [filename...]
strings지령은 2진파일에서ASCII문자렬을 찾는다.

문자렬은 행바꾸기문자나 빈 문자로 끝나는 4개이상의 인쇄문자들의 렬이다.

strings는 임의의 목적파일과 다른것을 식별하는데 사용된다.

실례 42

strings /bin/nawk | head -2

설명

실행가능한 2진파일 */bin/nawk*의 처음 2개의 행에 있는 ASCII본문을 인쇄한다.

stty-말단을 위한 추가선택설정

stty [-a] [-g] [modes]

stty는 현재표준입력인 장치에 대한 어떤 말단I/O추가선택들을 설정하며 인수가 없으면 어떤 추가선택들의 설정을 통보한다.

실례 43

1. stty erase <Press backspace key> or ^h
2. stty -echo; read secretword; stty echo
3. stty -a (AT&T) or stty -everything (BSD)

설명

1. 뒤걸음건을 지우기로 설정한다.
2. echoing을 해제하고 사용자입력을 대기한다. 그다음 echo행을 다시 설정한다.
3. stty에 대한 모든 가능한 추가선택들을 표시한다.

su-주사용자 또는 기타 사용자로 되기

su [-] [username [arg...]]

su는 등록탈퇴하지 않고 다른 사용자를 허용한다. 기정사용자이름은 root(주사용자)이다. su를 사용하려면 적당한 암호가 제공되여야 한다. 만약 통과문자렬이 정확하다면 su는 실질적이고 효과적인 사용자ID, 그룹ID, 지정된 사용자이름에 대한 보충적인 그룹목록모임을 가지는 새로운 쉘프로쎄스를 작성한다. 새로운 쉘은 사용자이름의 암호파일입력의 쉘마당에 지정된 쉘이다. 만일 쉘이 지정되지 않으면 sh(Bourne쉘)가 리용된다. 표준사용자ID에 우선권을 되돌리려면 Ctrl-D를 입력하여 새로운 쉘을 탈퇴한다. –추가선택은 완전한 등록가입을 지적한다.

sum-파일에 대한 검사합계산

sync-superblock를 갱신하고 변경된 블로크를 디스크에 보내기

tabs-말단에서 설정타브 정지

tail-파일의 끝현시

tail + [-number [lbc] [f] [filename]

tail + [-number [l] [rf] [filename]

+기호가 수자앞에 있을 때 tail은 블로크, 문자 또는 파일의 시작으로부터 계수하는 행들을 표시한다. 만일 련결부호(-)가 수자앞에 있을 때 tail은 파일의 마지막부터 계수한다.

실례 44

1. tail +50 filex
2. tail -20 filex
3. tail filex

설명

1. 행 50에서 시작하는 filex의 내용을 현시한다.
2. filex의 마지막 20개 행들을 현시한다.
3. filex의 마지막 10개 행들을 현시한다.

talk-다른 사용자와의 대화허가

talk username [ttyname]

talk는 말단에서부터 다른 사용자의 말단까지의 행들을 복사하는 시각적인 통신프로그람이다.

실례 45

talk joe@cowboys

설명

cowboy라는 콤퓨터에서 사용자 joe와 대화하기 위해 요구한다.

tar-문서파일, 보통 테프구동기로부터 파일들을 기억하거나 재생

tar [-] c|r|t|u|x [bBefFhilmopvwX0134778] [tarfile]
[blocksize] [exclude-file] [-I include-file]
filename1 filename2... -C directory filenameN...

실례 46

1. tar cvf / dev / diskette
2. tar tvf / dev / fd0
3. tar xvf / dev / fd0

설명

1. 현재작업등록부아래에 있는 모든 파일들을 장치 /dev/diskette에 있는 테프에 보내고 보내여 진 파일들을 인쇄한다.
2. 테프장치 /dev/fdo에 있는 내용들에 대한 표를 현시한다.

3. 모든 파일들을 테프에서 추출하고 어느 파일이 추출되였는가를 인쇄한다.

tee-표준출력을 복사

tee [-ai] [filename]

tee는 파일들이 ls|tee outfile에서와 같이 표준입력을 표준출력과 하나이상의 파일에 복사한다. 출력은 화면과 outfile로 간다.

실례 47

```
date | tee nowfile
```

설명

date지령의 출력이 현시장치에 현시되고 nowfile에 기억된다.

telnete-원격호스트와 통신

실례 48

```
telnet necom . com
```

설명

원격호스트 necom.com와 대화조종을 시작한다.

test-식평가

test는 식을 평가하고 그 식이 참(령)이거나 거짓(령아닌 값)이라는것을 가리키는 상태를 되돌린다. 기본적으로 문자렬, 수자, 파일시험을 위해서 Bourne와 Korn쉘에서 리용된다. C쉘은 대부분의 검사내부지령을 가지고 있다.

실례 49

```
1. test 5 gt 6
2. echo $? (Bourne and Korn Shells )
   (output is 1, meaning he result of the test is not true.)
```

설명

1. test지령은 옹근수검사를 진행하여 5가 6보다 큰가를 본다.
2. $?변수는 마지막지령의 탈퇴상태를 포함한다. 만일 령아닌 상태가 통보되면 검사결과는 참이 아니지만 되돌리는 상태가 령이면 검사결과는 참이다.

time-이 쉘과 그의 자식에 의하여 리용된 총시간 현시

timex-지령의 동기화; 프로쎄스자료와 체계활동상태통보

timex [-o] [-p] [-fhkmrt] [-s] command

주어 진 지령이 실행된다. 즉 경과된 시간, 사용자시간, 실행에 소비된 체계시간은

초단위로 통보된다. 선택적으로 지령에 대한 자료를 계산하는 프로쎄스와 모든 그 자식 프로쎄스들이 표시되거나 합계될수 있다. 실행기간에 체계활동상태가 통보될수 있다. timex의 출력은 표준오유에 씌여 진다.

touch-파일의 호출시간이나 변경시간 갱신

touch [-amc] [mmddhhmm [yy]] filemame...

touch는 매 인수의 호출시간과 변경시간을 갱신되게 한다. 파일이름은 존재하지 않으면 작성된다. 시간이 지정되지 않으면 현재시간이 리용된다.

실례 50

```
touch a b c
```

설명

3개의 파일들인 a, b, c가 작성된다. 그것들중 임의의 어떤 파일이 존재하면 그 파일의 변경시간이 갱신된다.

tput-말단초기화 또는 terminfo자료기지를 질문

tput [-Ttype] capname [parms...]
tput [-Ttype] init
tput [-Ttype] reset
tput [-Ttype] longname
tput -S < <

tput는 terminfo자료기지를 리용하여 말단의존능력값과 쉘에서 리용할수 있는 정보가 말단을 초기화 또는 재설정하게 한다. 또는 요구되는 말단형의 긴 이름을 되돌린다.

실례 51

```
1. tput longname
2. bold= · put smso ·
   unbold='tput rmso'
   echo "${bold} Enter your id: $(offbold) \ c "
```

설명

1. terminfo자료기지로부터 말단의 긴 이름을 현시한다.
2. 쉘변수 bold를 설정하여 현시된 본문을 강조한다. 그다음 쉘변수 unbold를 설정하여 보통본문을 현시한다. 행 Enter your id:은 검은색바탕에 흰 문자로 강조된다. 그다음 본문은 보통방식으로 현시된다.

tr-문자들의 변환

tr. [-CDs] [string1 [string2]]

tr는 표준입력을 선택된 문자로 바꿔넣기 또는 소거하여 표준출력에 복사한다. string1에 있는 입력문자는 string2의 대조하는 문자들로 넘긴다. 빗선은 8진수와 함께

리용되여 ASCII코드를 표시한다. string2물음표수가 string1보다 적으면 string2에 대조하는 문자를 가지고 있지 않는 string1물음표들은 번역되지 않는다. 문자들에 대한 8진수값은 그 앞에 거꿀빗선이 있을 때 리용할수 있다.

\ 11　타브
\ 12　행바꾸기문자
\ 042　단일인용부호
\ 047　2중인용부호

실례 52

1. tr. ‘A’ ‘B‘< filex
2. tr. ‘[A-Z]’ ‘[a-z]’< filex
3. tr. -d ‘ ‘ < filex
4. tr. -s ‘\ 11’ ‘\ 11’ < filex
5. tr. -s ‘:’ < filex
6. tr. ‘\ 047’ ‘\ 042’

설명

1. filex에서 A를 B로 변환한다.
2. 모든 대문자들을 소문자로 변환한다.
3. filex에서 모든 공백들을 삭제한다.
4. filex에서 다중타브를 단일타브로 교체한다.
5. filex에서 다중옹근두점을 단일공백으로 교체한다.
6. 표준입력에서 나오는 본문에서 2중인용부호를 다중인용부호로 변환한다.

true-성공적인 탈퇴상태의 제공

true는 아무것도 하지 않지만 성공하면 항상 령을 되돌린다. Bourne와 Korn쉘프로그람에서 무환순환을 시작하는 지령으로 사용한다.

```
while true
do
  command
done
```

tsort-위치적정렬

/ usr /c.c.s /bin / tort [filename]

tsort지령은 표준출력을 만들며 입력파일에서 언급된 항목들의 부분적인 순서로 이루어 진 순서화된 목록을 만든다. 파일이름이 지적되지 않으면 표준입력은 해석된다. 입력은 공백으로 분리되여 있는 항목들의 쌍으로 이루어 진다. 서로 다른 항목들의 쌍은 순서를 표시한다. 동일한 항목들의 쌍은 존재를 표시하지만 순서는 표시하지 않는다.

tty-말단의 이름얻기

tty [-l] [-s]

tty는 사용자말단의 경로이름을 인쇄한다.

unmask-허가를 위한 파일창조방식마스크 설정

unmask [000]

사용자파일작성방식마스크는 000으로 설정된다. 3개의 8진수는 소유자, 그룹, 기타에 대한 읽기/쓰기/실행허가를 표시한다. 지적된 매 수자의 값은 파일작성을 위해 체계에서 지정된 대조하는 《수자》로부터 던다. 실례로 umask 022는 그룹과 다른것의 쓰기허가를 지우기한다(방식 777로 작성된 파일이 방식 755로 되며 방식 666으로 작성된 파일이 644로 된다.). 만일 000이 생략되면 마스크의 현재값이 인쇄된다. umask는 쉘에 의해 인식되고 실행된다.

실례 53

1. umask
2. umask 027

설명

1. 현재파일허가마스크를 현시한다.
2. 등록부허가 777에서 umask 027를 덜면 750이다. 파일허가 666에서 umask 027을 덜면 640이다. 등록부와 파일들이 작성될 때 umask에 의하여 작성된 허가가 대입된다.

uname-현재콤퓨터의 이름을 인쇄

uname [-amnprsv]
uname [-S system_name]

uname은 현재체계에 대한 정보를 표준출력에 인쇄한다. 만일 추가선택들이 지정되지 않으면 uname은 현재조작체계의 이름을 인쇄한다. 추가선택들은 uname과 sysinfo로 되돌려 지는 선택된 정보를 인쇄한다.

실례 54

1. uname -n
2. uname -a

설명

1. 주콤퓨터의 이름을 인쇄한다.
2. -m, -n, -r, -s, -v와 같은 콤퓨터의 하드웨어이름, 망접속이름, 조작체계 계렬번호, 조작체계이름, 조작체계판본을 인쇄한다.

uncompress-압축지령을 리용하여 파일들이 압축된후 그것들을 초기상태로 회복

uncompress [-cFv] [file...]

실례 55

uncompress file.Z

설명

file.Z를 그의 초기상태 즉 그 파일이 압축되기전 상태로 다시 회복한다.

uniq-파일에 있는 중복행들을 통보

uniq [[-u] [-d] [+n] [-n]] [input [output]]

uniq는 린접행들과 비교하면서 입력파일들을 읽는다. 보통 반복행들의 두번째이상의 사본들이 지우기된다. 나머지는 출력파일에 씌여 진다. 입력와 출력은 언제나 차이나야 한다.

실례 56

1. uniq file1 file2
2. uniq -d -2 file3

설명

1. file1로부터 중복되는 린접행들을 지우기하고 출력을 file1에 넣는다.
2. 중복은 세번째 마당에서 시작되는 중복행들을 현시한다.

units-표준척도로 나타낸 량들을 다른 척도로 변환

units는 여러가지 표준척도로 나타낸 량들을 다른 척도로 변환한다. 이것은 이 방식에서 대화형으로 작용한다.

```
You have: inch
You want:cm
   *2.540000e+00
   /3.937008e-01
```

unpack-pack로 창조된 파일을 확장

unpack는 pack로 작성된 파일을 확장한다. 지령에서 지정된 매 파일이름에 대하여 탐색은 name.z라는 파일을 위해서 진행된다. 만일 이 파일이 묶음파일로 나타나면 그것은 확장된 판본으로 교체된다. 새로운 파일은 자기 이름에서 지우기된 뒤붙이 .z를 가지며 묶음파일과 같은 호출방식, 호출날자, 변경날자와 소유자를 가진다.

uucp-또 다른 체계에 파일을 복사, UNIX와 UNIX사이의 체계복사

uucp [-c|-C] [-d|-f] [-ggrade] [-j] [-m] [-nuser] [-r] [-sfile] [-xdebug _level]

uucp는 원천파일인수들로 이름 지어 진 파일들을 목적파일인수에 복사한다.

uuencode-uuencode, uudecode-2진파일을 전자우편을 통하여 보내기 위해 ASCII본문으로 부호화하거나 초기형식으로 변환

uuencode [sorce-file] file-label
uudecode [encode-file]

uuencode는 2진파일을 우편을 통하여 보낼수 있는 ASCII코드로 변화시킨다. 표식인수는 복호화할 때 리용하는 출력파일이름을 지정한다. 만일 파일이 주어 지지 않으면 stdin을 부호화한다. uudecode는 부호화된 파일을 읽고 발송자프로그람에 의해서 추가된 머리부와 꼬리부행들을 없애며 머리부에서 지적된 파일이름방식 그리고 소유자로 초기 2진자료를 다시 만든다. 부호화된 파일은 보통 ASCII본문파일이며 그것은 임의의 본문편집기로 편집될수 있다. 그러나 복호화된 2진파일의 파괴를 막으려면 머리부에 있는 방식이나 파일표식을 변화시키는것이 제일 좋다.

실례 57

1. uuencode mybinfile decodedname > uumybinfile . tosend
2. uudecode uumybinfule . tosend

설명

1. 첫 인수 mybinfile은 부호화해야 할 현존파일이다. 두번째 인수는 파일을 전송한후 uuencoded파일을 위하여 리용되는 이름이며 uuybinfile.tosend는 우편을 통하여 전송되는 파일이다.
2. 이것은 uuencode파일을 복호화하고 uuencode에 대한 두번째 인수로 주어질 때 파일이름을 작성한다.

wc-행, 단어, 문자를 계수

wc [-lwc] [filename...]

wc는 파일이름이 주어 지지 않으면 파일이나 표준입력에서 행, 단어, 문자들을 계수한다. 단어는 공백, 타브, 행바꾸기문자로 분리되는 문자렬이다.

실례 58

1. wc filex
2. who | wc -l
3. wc -l filex

설명

1. filex에서 행, 단어, 문자들의 수를 인쇄한다.
2. who지령의 출력은 wc로 파이프되여 계수되는 행수를 표시한다.
3. filex의 행수를 표시한다.

what-@(#)패턴뒤에 있는 정보를 인쇄하여 파일로부터 SCCS판본정보얻기

what [-s] filename

what는 파일이름을 탐색하여 패턴 @(#)을 찾는데 SCCS get지령은 그것으로 %Z%예약어를 교체하고 “>, 행바꾸기문자, \ 또는 빈 문자까지 뒤따르는 모든것을 인쇄한다.

which-지령을 찾고 그의 경로이름이나 별명들을 현시

which [filename]

which는 이름들의 목록을 가지고 이름이 인수로 주어 지게 하며 실행되는 파일들을 찾는다. 매 인수는 별명이면 확장되므로 사용자의 경로를 따라서 탐색된다. 별명들과 경로는 둘다 사용자의 .cshrc파일에서 읽는다. .cshrc파일만이 리용된다.

whereis –지령에 대한 2진파일, 원천파일, 수동페지파일찾기(UCB)

whereis [-bmsu] [-BMS directory ... -f] filename

who-체계에 등록가입된 사용자를 현시

write-통보문을 다른 사용자에게 쓰기

write username [ttyname]

write는 말단으로부터 다른 사용자의 말단장치로 행들을 복사한다.

xrags-인수들의 목록을 만들고 지령을 실행

xrags [flags] [command] [Initial-arguments]]

xrags는 파일의 내용들을 지령행으로 보내고 동적으로 지령행들을 만들게 한다.

실례 59

```
1. ls $1 | xrags -i -t mv $1/{} $2/{}
2. ls | xrags -p -l rm -rf
```

설명

1. 등록부 $1로부터 등록부 $2까지 모든 파일들을 이동시키므로 매 mv지령을 실행하기전에 출력한다.
2. 사용자에게 한번에 하나씩 지우기되는 파일들을 재촉하고 그것들을 지우기 한다.

zcat-압축된 파일을 표준출력에로 풀기. 다음의 zcat지령은 uncompress-c와 같다.

zcat [file...]

실례 60

```
zcat book . doc . z | more
```

설명

book.doc.z를 확장하고 그 출력을 more에 파이프한다.

부록 2. 쉘들의 비교

1. 비교하는 쉘

특징	Bourne	C	T	Korn	Bash
별명	없음	있음	있음	있음	있음
개선된 패턴대조	없음	없음	없음	있음	있음
지령행편집	없음	없음	있음	있음*	있음
등록부탄창	없음	있음	있음	없음	있음
파일이름완성	없음	있음*	있음	있음	있음
함수	있음	없음	없음	있음	있음
리력	없음	있음	있음	있음	있음
일감조종	없음	있음	있음	있음	있음
건??	없음	없음	있음	없음	있음
재촉문형식화	없음	없음	있음	없음	있음
맞춤법교정	없음	없음	있음*	없음	있음†

* 기정으로 설정되지 않으면 사용자가 설정하여야 한다.
† cdspell은 cd를 사용할 때 등록부이름에서 적은 맞춤법오유을 바로 잡는데 설정하는 shop추가선택이다.

2. tcsh와 csh

TC쉘(tsch)은 Berkely C쉘의 개선된 판본이다. 여기서 표시된것들은 새로운 특징들이다.

- 강화된 리력기구
- 지령행을 편집하기 위한 내부지령행편집기(emacs 또는 vi)
- 재촉문들을 형식화
- 맞춤법수단과 맞춤법과 순환을 위한 특수재촉문
- 지령, 파일이름, 변수, 사용자이름 등을 완성하기 위한 강화되고 프로그람화된 단어완성기능
- 건맺기들을 작성하고 수정하는 능력
- 자동적이고 정기적이고 시간적인 사건(일정작성사건, 특별한 별명, 자동등록탈퇴, 말단열쇠걸기 등)
- 새로운 내부지령들(hup, ls-F, new grep, print env, which, where 등)
- 새로운 내부변수들(gid, loginsh, oid, shlvl, tty, version, HOST, REMOTEHOS-T, VENDOR, OSTYPE, MACHTYPE)
- 읽기전용변수
- 보다 좋은 오유통보수단

3. bash 와 sh

Bourne Again쉘은 전통적인 Bourne쉘에는 없는 다음의 특징을 가지고 있다.

- 재촉문의 형식화
- 리력(csh형식)
- 별명들
- 지령행편집을 위한 내부지령행편집기(emacs 또는 vi)
- pushd와 popd를 리용한 등록부조작
- 배경에서 일감들을 정지시키거나 실행시켜 그것들을 전경으로 가져 오는 csh형일감조종. 이때 bg, fg, Ctrl-Z 등과 같은 지령을 리용한다.
- 물결표, 대괄호 그리고 파라메터의 확장
- 건문자렬들을 전용화하는 건맺기
- 고급한 패턴대조
- 배렬
- select순환(Korn쉘로부터)
- 많은 새로운 내부지령들

특징	csh/fcsh	Bourne	Bash	Korn
변수:				
국부변수에 값대입	set x = 5	x=5	x=5	x=5
변수속성대입			declare or typeset	typeset
환경변수에 값대입	Setenv NAME Bob	NAME=' Bob' ; export NAME	export NAME='Bob'	export NAME='Bob'
읽기전용변수:				
변수호출	echo $NAME set var = net echo $(var)work network	echo $NAME var=net echo ${var}work network	echo $NAME var=net echo ${var}work network	echo $NAME or print $NAME var=net echo ${var}work network
문자개수	echo $%var (tcsh only)	N/A	${#var}	${#var}
특수변수:				
프로쎄스의 PID	$$	$$	$$	$$
탈퇴상태	$status, $?	$?	$?	$?
마지막배경일감	$! (tcsh only)	$!	$!	$!

특징	csh/fcsh	Bourne	Bash	Korn
배열:				
배열대입	set x = (a b c)	N/A	y[0]='a'; y[2]='b'; y[2]='c' fruit=(apples pears peaches plums)	y[0]='a'; y[2]='b'; y[2]= 'c' set –A fruit apples pears plums
배열요소호출	echo $x[1] $x[2]	N/A	echo ${y[0]} ${y[1]}	print ${y[0]} ${y[1]}
전체 요소	echo $x or $x[*]	N/A	echo ${y[*]}, ${fruit[0]}	print ${y[*]}, ${fruit[0]}
요소번호	echo $#x	N/A	echo $y{#[*]}	print ${#y[*]}
지령대입:				
지령출력을 변수에 대입	set d = \`date\`	d = \`date\`	d=$(date) or d = \`date\`	d=$(date) or d = \`date\`
값호출	echo $d echo $d[1],$d[2], … echo $#d	echo $d	echo $d	print $d
지령행인수(위치파라메터):				
호출	$argv[1],$argv[2] or $1, $2...	$1,$2, ... $9	$1, $2, ... ${10}...	$1, $2, ... ${10}...
위치파라메터의 설정	N/A	set a,b,c set \`date\` echo $1 $2 ...	set a,b,c set \`date\` or set $(date) echo $1 $2 ...	set a,b,c set \`date\` or set $(date) printf $1 $2...
지령행인수번호	$#argv $# (tcsh)	$#	$#	$#
$arg[number에서 문자번호	$%1,$%2 (tcsh)	N/A	N/A	N/A
파일이름확장메타문자:				
단일문자	?	?	?	?
령개이상문자	*	*	*	*
모임에서 한 문자	[abc]	[abc]	[abc]	[abc]
모임에 있는 문자 범위의 한 문자	[a-c]	[a-c]	[a-c]	[a-c]

특징	csh/fcsh	Bourne	Bash	Korn
모임에서 없는 한문자	N/A(csh) [^abc] (tcsh)	[!abc]	[!abc]	[!abc]
? 는 괄호안에 있는 패턴물음표를 대신하거나 아무것도 대신하지 않는다. 수직선은 or 조건을 표시한다. 실례로 2 또는 9, abc21, abc91 또는 abcl. 파일이름은 패턴 ^patt~ern(tcsh)과 대조하지 않는다.			Abc?(2|9)1	abc?(2|9)1
I/O 방향바꾸기와 파이프:				
지령출력을 파일에로 방향바꾸기	cmd > file	cmd > file	cmd > file	cmd > file
지령출력을 파일에 방향바꾸기하고 추가	cmd >> file	cmd >> file	cmd >> file	cmd >> file
지령입력을 파일로부터 방향바꾸기	cmd < file	cmd < file	cmd < file	cmd < file
지령오유를 파일에로 방향바꾸기	(cmd > /dev/tty)& errors	cmd 2>errors	cmd 2>file	cmd 2>errors
지령출력과 오유를 파일에로 방향바꾸기	cmd >& file	cmd > file 2>&1	cmd >& file or cmd &> file or cmd > file 2>&1	cmd > file 2>&1
출력대입과 noclobber를 무시	cmd >|file	N/Ac	cmd >| file	cmd >| file
here문서	cmd << EOF input EOF	cmd << EOF input EOF	cmd << EOF input EOF	cmd << EOF input EOF
한개지령의 출력을 다른 지령의 입력에 파이프	cmd | cmd	cmd | cmd	cmd | cmd	cmd | cmd
출력과 오유를 지령에 파이프하기	cmd |& cmd	N/A	N/A	(See coprocesses)
협동프로쎄스	N/A	N/A	N/A	command |&
조건명령문	cmd && cmd cmd || cmd	cmd && cmd cmd || cmd	cmd && cmd cmd || cmd	cmd && cmd cmd || cmd

특징	csh/fcsh	Bourne	Bash	Korn
건반으로부터 읽기:				
변수에 입력행읽기와 기억	set var = $< set var = \`line\`	read var read var1 var2…	read var read var1 var2… read read –p prompt read –a arrayname	read var read var1 var2… read read var?"Enter value"
산수연산:				
계산	@ var = 5 + 1	var=\`expr 5 + 1\`	((var = 5 + 1))	((var = 5 + 1))
물결표확장:				
사용자홈등록부표시	~username	N/A	~username	~username
홈등록부표시	~	N/A	~	~
현재작업등록부표시	N/A	N/A	~+	~+
이전작업등록부표시	N/A	N/A	~–	~–
별명:				
별명만들기	alias m more	N/A	alias m=more	alias m=more
별명목록제시	alias		alias, alias –p	alias, alias –t
별명지우기	unalias m	N/A	unalias m	unalias m
리력:				
리력설정	set history = 25	N/A	automatic or HISTORY=25	automatic or HISTORY=25
번호가 있는 리력목록현시	history		history, fc –1	history, fc –1
번호로 선택된 목록부분현시	history 5		history 5	history 5 10
지령의 재실행	!! (마지막지령) !5 (다섯번째 지령) !v (v로 시작하는 마지막지령)		!! (마지막지령) !5 (다섯번째 지령) !v (v로 시작하는 마지막지령)	r (마지막지령) !5 (다섯번째 지령) r v (v로 시작하는 마지막지령)
대화형편집기설정	N/A(csh) bindkey –v 또는 bindkey –e(tcsh)	N/A	set –o vi set –o emacs	set –o vi set –o emacs
신호:				
지령	onintr	trap	trap	trap

특징	csh/fcsh	Bourne	Bash	Korn
초기화파일:				
등록가입시 실행	.login	.profile	.bash_profile	.profile
쉘이 호출될 때마다 실행	.cshrc	N/A	BASH_ENV=.bashrc (or other filename)(bash 2.x) ENV=.bashrc	ENV=.kshrc (or other filename)
함수:				
함수정의	N/V	fun() { commands; }	function fun { commands; }	function fun { commands; }
함수호출	N/V	fun	fun	fun
		fun param1 param2...	fun param1 param2...	fun param1 param2...
프로그람작성구조:				

if 조건	if (식)
	then
	commands
	endif
	if
	{ (commands) }
	then
	commands
	endif
		if (식)
		then
		commands
		fi
		if
		{ (commands) }
		then
		commands
		endif
			if [[문자렬식]]
			then
			commands
			fi
			if ((수식))
			then
			commands
			fi
				if [[문자렬식]]
				then
				commands
				fi
				if ((수식))
				then
				commands
				fi

if/else조건	if (식)
	then
	commands
	else
	commands
	endif
		if command
		then
		commands
		else
		...
		fi
			if command
			then
			commands
			else
			...
			fi
				if command
				then
				commands
				else
				...
				fi

if/else/elseif 조건	if (식) then
	commands
	else if (식) then
	commands
	else
	commands
	endif
		if command
		then
		commands
		elif
		commands
		else
		commands
		fi
			if command
			then
			commands
			elif
			commands
			else
			commands
			fi
				if command
				then
				comands
				elif
				commands
				else
				commands
				fi

goto	goto label ... label:	N/V	N/V	N/V

특징	csh/fcsh	Bourne	Bash	Korn
switch 와 case	switch ("$value") case pattern1: commands breaksw case pattern2: commands breaksw default: commands breaksw endsw	case "$value" in pattern1) commands ;; pattern2) commands ;; *) commands ;; esac	case "$value" in pattern1) commands ;; pattern2) commands ;; *) commands ;; esac	case "$value" in pattern1) commands ;; pattern2) commands ;; *) commands ;; esac
순환:				
while 순환	while (식) commands end	while command do command done	while command do command done	while command do command done
for/foreach	foreach var (단어목록) commands end	for var in 단어목록 do commands done	for var in 단어목록 do commands done	for var in 단어목록 do commands done
until		until command do commands done	until command do commands done	until command do commands done
repeat	repeat 3 "echo hello" hello hello hello	N/V	N/V	N/V
slect	N/V	N/V	PS3="Please slect a menu item" slect var in단어목록 do commands done	PS3="Please slect a menu item" slect var in단어목록 do commands done

부록 3. 인용부호를 정확히 리용하기 위한 걸음들

1. 거꿀빗선

① 문자앞에 붙여 그 문자를 특수한 의미에서 벗어 나게 한다.
② 한개 문자주위에 단일인용부호를 쓰는것과 같다.

2. 단일인용부호

① 대조되여야 한다.
② 다음의것을 제외하고는 모든 메타문자들이 해석되지 않게 한다.

ㄱ. 자기자체
ㄴ. 감탄부호(csh에서만)
ㄷ. 거꿀빗선

실례:

C쉘	Bourne쉘	Korn쉘
echo '$><%^&*'	echo '$*&!><?'	pritnt '$*&!><'
echo "I need $5.00\ "	echo "I need $5.00\ "	pritnt "I need $5.00\ "
echo 'She cried,"Help"'	echo 'She cried,"Help" '	pritnt 'She cried,"Help"'
echo '\ \ \ \ '	echo '\ \ \ \ '	pritnt '\ \ \ \ '
\ \ \ \	\ \	\ \

3. 2중인용부호

① 대조되여야 한다.
② 다음의것을 제외하고는 모든 메타문자들이 해석되지 않게 한다.

ㄱ. 자기자체
ㄴ. 감탄부호(csh에서만)
ㄷ. 변수바꿔넣기를 위해 리용되는 화페기호
ㄹ. 지령바꿔넣기에 리용되는 거꿀인용부호

실례 :

C쉘	Bourne쉘	Korn쉘
echo "Hello $LOGNAME\ !"	echo "Hello $LOGNAME!"	print "Hello $LOGNAME!"
echo "I don't care"	echo "I don't care"	print "I don't care"
echo "The date is `date`"	echo "The date is `date`"	print "The date is `date`"
echo "\ \ \ \ "	echo "\ \ \ \ "	print "\ \ \ \ "
\\\\	\	\

4. 인용부호조합

마지막결과는 쉘변수를 awk지령행에 매몰할수 있으며 쉘이 nawk의 마당지시자 $1과 $2를 해석하지 않고 변수를 확장하게 한다.

5. 쉘변수설정

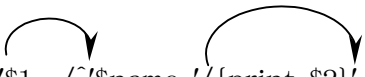
name=”Jacob Savage” (Bourne and Korn Shell)
set name = “Jacob Savage” (C shell)

(The line from the datafile)
Jacob Savage:408-298-7732:934 La Barbara Dr.,San Jose, CA:02/27/78:50000

(The nawk command line)
nawk –F: `$1 ~/^` “$name” `/{print $2}` datafile

(Output)
408-298-7732

단계 1: 쉘변수에 대입하기전에 지령행에서 UNIX지령을 확인한다.

nawk –F:$1 ~/^Jacob Savage/{print $2}` filename

(Output)
408-298-7732

단계 2: 그 밖의 임의의것을 변화시키지 않고 쉘변수에 대입한다. 모든 인용부호들을 그대로 남겨 놓는다.

nawk –F: `$1~/^$name/{print $2}` datafile

awk지령의 왼쪽에서 시작하여 첫번째 인용부호를 남겨 놓는다. $name에 있는 쉘화페기호앞에 다른 인용부호를 넣는다. 이때 첫번째 인용부호는 대조되고 이 2개의 인용부호안에 있는 모든 본문은 쉘해석으로부터 보호된다. 변수는 로출된다. 이때 $name에 있는 e뒤에 또 다른 단일인용부호를 넣는다. 이렇게 하면 이것은 awk의 닫는 대괄호뒤에 있는 단일인용부호모임과 대조된다. 이 인용부호쌍안에 있는 모든것이 쉘해석으로부터 보호된다.

nawk –F: '$1 ~/^'$name '/{print $2}' datafile

단계 3: 한쌍의 2중인용부호안에 쉘변수들을 넣는다. 이렇게 하면 변수는 확장되지만 변수의 값은 공백을 포함하고 있을 때 단일문자렬로 취급된다. 지령행이 정확히 해석되도록 공백이 보호되여야 한다.

인용부호의 수를 계수한다. 단일인용부호의 수와 2중인용부호의 수는 우수로 되여야 한다.

실례:

oldname="Ellie Main"
newname="Eleanor Quigley"

1. 지령이 실행되게 한다.

   nawk –F: '/^Ellie main{$1={Eleanor Quigley";print $0}'datafile

2. 변수들을 대입한다.

   Nawk –F:'/^$oldname/{$1="$newname";print $0}' datafile

3. 인용부호찍기를 진행한다. 왼쪽에 있는 첫번째 단일인용부호에서 시작하여 수 $oldname전까지 선을 긋고 $앞에 다른 단일인용부호를 넣는다. 변수이름의 마지막문자뒤에 또 다른 단일인용부호를 넣는다. 이때 오른쪽으로 이동하여 변수 $newname에 있는 화페기호앞에 단일인용부호를 또 넣는다. $new name의 마지막문자뒤에 단일인용부호를 넣는다.

4. 단일인용부호의 수를 계수한다. 그 개수가 우수이면 매 단일인용부호들은 대조된다. 우수가 아니면 한 단계를 뛰여 넘은것이다. 모든 쉘변수들을 2중인용부호안에 넣는다. 쉘변수주위에 2중인용부호를 적당히 넣는다.

색 인

ㄱ

ㄴ

ㄷ

ㄹ

ㅁ

ㅂ

ㅅ

ㅈ

ㅊ

ㅌ

ㅍ

* * * * *

A

B

C

D

I

J, k

L

N, M

O

P

Q, R

S

T

U

V

W

X, Y

* * * * *